módulo 1

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 1

O objetivo desta lista de exercícios é você praticar algumas operações matemáticas básicas, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Parte 1 - Operações Básicas

Exercício 1 - Leia 2 números do terminal e imprima a soma, subtração, multiplicação e divisão.

Dica: Crie funções em Python para formatar a saída das operações

In [1]:

Nome do função

```
# Solução

def add(a, b):
    print('\nResultado da soma entre {0} e {1} é {2} '.format(a, b, a+b))

def subtract(a, b):
    print('\nResultado da soma subtração entre {1} e {0} é {2}'.format(a, b, a-b))

def divide(a, b):
    print('\nResultado da soma divisão entre {0} e {1} é {2}'.format(a, b, a/b))

def multiply(a, b):
    print('\nResultado da multiplicação entre {0} e {1} é {2}'.format(a, b, a*b))
```

soma

```
try:
```

tenta executar

```
    a = int(input('Digite o primeiro número: '))
    b = int(input('Digite o primeiro número: '))

    add(a, b)
    subtract(a, b)
    divide(a, b)
    multiply(a, b)

except ValueError:
```

entrada com valor incorreto

```
    print('Operação não permitida')
except ZeroDivisionError:
    print('Operação não permitida')
```

```
Digite o primeiro número: 12
Digite o primeiro número: 4

Resultado da soma entre 12 e 4 é 16

Resultado da soma subtração entre 4 e 12 é 8

Resultado da soma divisão entre 12 e 4 é 3.0

Resultado da multiplicação entre 12 e 4 é 48
```

Exercício 2 - Escreva um programa em Python para converter uma medida de graus em radianos.

Nota: O radiano é a unidade padrão de medida angular, usada em muitas áreas da matemática. A medida de um ângulo em radianos é numericamente igual ao comprimento de um arco correspondente de um círculo unitário; um radiano é um pouco menos que 57,3 graus (quando o comprimento do arco é igual ao raio).

In [2]:

```
# Solução

pi = 22/7 # (3.14)
graus = float(input("Digite a medida em graus: "))
radiano = graus * (pi/180)
print(radiano)
```

```
Digite a medida em graus: 35
0.6111111111111112
```

Exercício 3 - Escreva um programa em Python para calcular a área de um paralelogramo.

Nota: Um paralelogramo é um quadrilátero com lados opostos paralelos (e portanto ângulos opostos iguais). Um quadrilátero com lados iguais é chamado de losango, e um paralelogramo cujos ângulos são todos ângulos retos é chamado de retângulo.

In [3]:

```
# Solução

base = float(input('Comprimento da base: '))
altura = float(input('Medida da altura: '))
area = base * altura
print("A área é: ", area)
```

```
Comprimento da base: 24
Medida da altura: 14
A área é:  336.0
```

Exercício 4 - Escreva um programa em Python para calcular o volume da superfície e a área de uma esfera.

Nota: Uma esfera é um objeto geométrico perfeitamente redondo no espaço tridimensional que é a superfície de uma bola completamente redonda.

In [4]:

```
# Solução

pi = 22/7 # (3.14)
radian = float(input('Radianos da esfera: '))
sur_area = 4 * pi * radian **2
volume = (4/3) * (pi * radian ** 3)
print("Área da Superfície: ", sur_area)
print("Volume: ", volume)
```

```
Radianos da esfera: 10
Área da Superfície:  1257.142857142857
Volume:  4190.47619047619
```

Exercício 5 - Escreva um programa em Python para imprimir todas as permutações de uma determinada string (incluindo duplicatas). Por exemplo a string: ABCD.

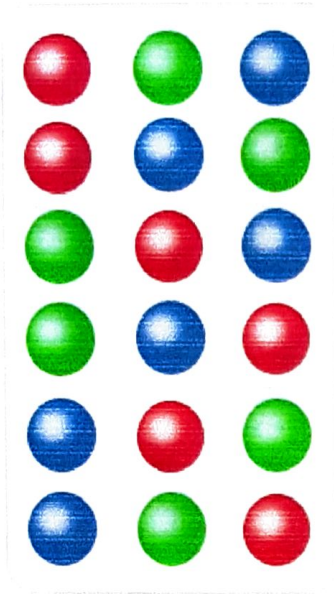
Na matemática, a noção de permutação refere-se ao ato de organizar todos os membros de um conjunto em alguma sequência ou ordem, ou se o conjunto já está ordenado, reorganizando (reordenando) seus elementos, um processo chamado de permutação. Estes diferem das combinações, que são seleções de alguns membros de um conjunto onde a ordem é desconsiderada.

Na imagem a seguir, cada uma das seis linhas é uma permutação diferente de três bolas distintas

In [5]:

```
from IPython.display import Image
Image("images/image02.png")
```

Out[5]:



In [6]:

```
def permute_string(str):
    if len(str) == 0:
        return ['']

    prev_list = permute_string(str[1:len(str)])
    next_list = []

    # Percorrendo a matriz formada pelas permutações
    for i in range(0,len(prev_list)):
        for j in range(0,len(str)):
            new_str = prev_list[i][0:j]+str[0]+prev_list[i][j:len(str)-1]
            if new_str not in next_list:
                next_list.append(new_str)
    return next_list

print(permute_string('ABCD'));
```

Recursividade chamo o método

de 1 até str = 4 = ABCD

0 → 4

adiciona em next_list

```
['ABCD', 'BACD', 'BCAD', 'BCDA', 'ACBD', 'CABD', 'CBAD', 'CBDA', 'ACDB', 'CA
DB', 'CDAB', 'CDBA', 'ABDC', 'BADC', 'BDAC', 'BDCA', 'ADBC', 'DABC', 'DBAC',
'DBCA', 'ADCB', 'DACB', 'DCAB', 'DCBA']
```

Exercício 6 - Escreva um programa em Python para calcular o valor discriminante.

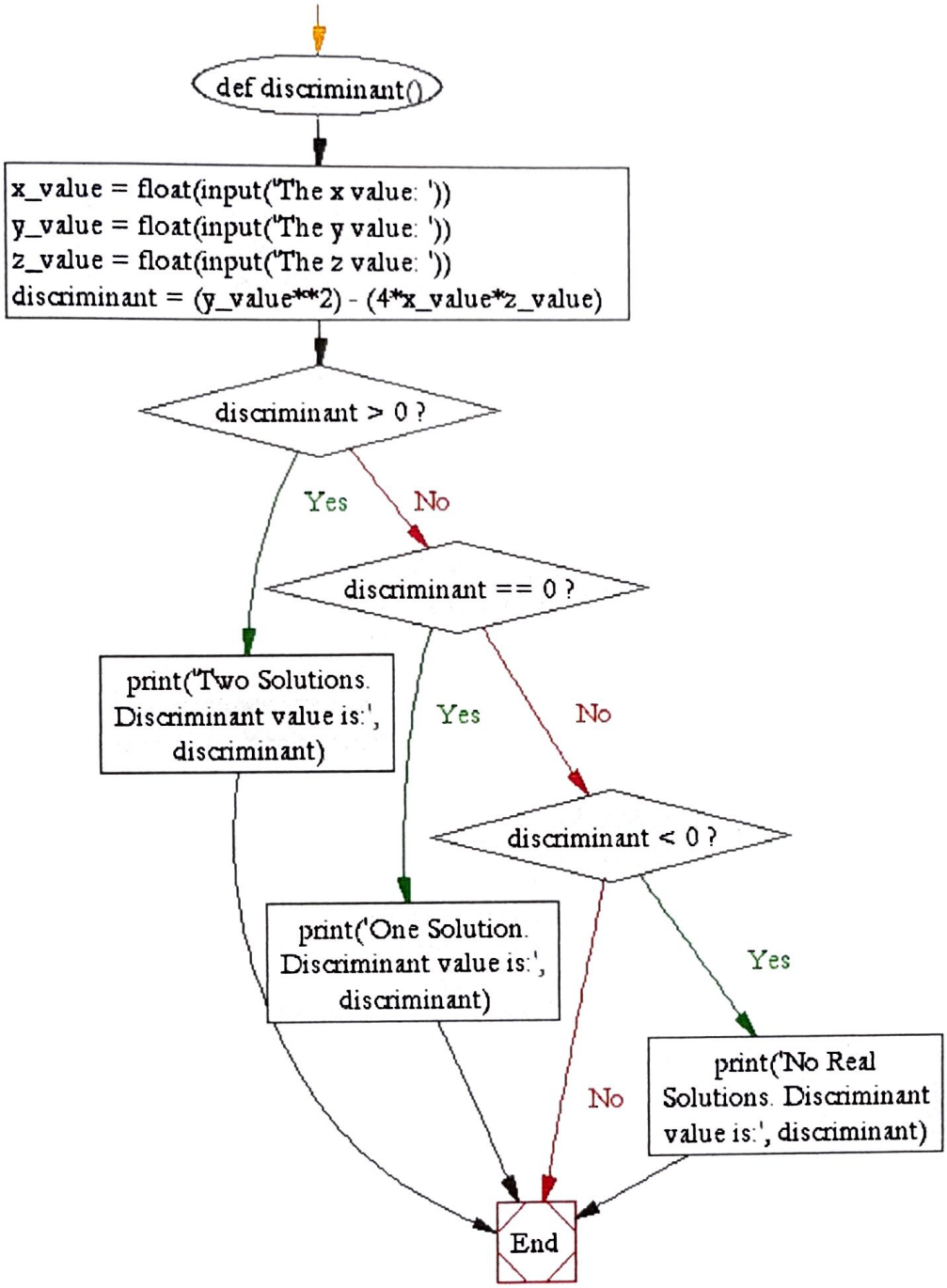
Nota: O discriminante é o nome dado à expressão que aparece sob o sinal de raiz quadrada (radical) na fórmula quadrática (amplamente usada em Machine Learning).

Dica: Crie uma função em Python com base na regra abaixo

In [7]:

```
from IPython.display import Image
Image("images/image01.png")
```

Out[7]:



In [8]:

```python
# Solução

def discriminant():
    x_value = float(input('Valor de x: '))
    y_value = float(input('Valor de y: '))
    z_value = float(input('Valor de z: '))

    discriminant = (y_value**2) - (4*x_value*z_value)

    if discriminant > 0:
        print('Duas Soluções. Valor discriminante é:', discriminant)
    elif discriminant == 0:
        print('Uma Solução. Valor discriminante é:', discriminant)
    elif discriminant < 0:
        print('Sem Solução Real. Valor discriminante é:', discriminant)


discriminant()
```

```
Valor de x: 1
Valor de y: 9
Valor de z: 6
Duas Soluções. Valor discriminante é: 57.0
```

Exercício 7 - Escreva um programa em Python para calcular raízes quadradas usando o método babilônico.

Talvez o primeiro algoritmo usado para se aproximar de √S seja conhecido como método babilônico, batizado em homenagem aos babilônios, ou "Hero's method", em homenagem ao matemático grego do primeiro século, Hero de Alexandria, que deu a primeira descrição explícita do método.

Pode ser derivado (mas antecede por 16 séculos) do método de Newton. A ideia básica é que, se x é uma superestimativa para a raiz quadrada de um número real não negativo S, então S/x será subestimado e, portanto, pode-se esperar que a média desses dois números forneça uma melhor aproximação.

Representação gráfica do método babilônico
https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#/media/File:Babylonian_method.svg
(https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#/media/File:Babylonian_method.svg)

In [9]:

```
# Solução

def BabylonianAlgorithm(number):
    if(number == 0):
        return 0;

    g = number/2.0;
    g2 = g + 1;
    while(g != g2):
        n = number/ g;
        g2 = g;
        g = (g + n)/2;

    return g;

print('A raiz quadrada de 9 é ', BabylonianAlgorithm(9));
```

```
A raiz quadrada de 9 é  3.0
```

Exercício 8 - Escreva um programa em Python que receba um arquivo com notas de um teste e divida as notas em classes de frequência.

Dica: seu programa deve permitir ao usuário escolher o número de classes e fazer a leitura do arquivo

In [10]:

```python
# Solução

def create_classes(numbers, n):
    low = min(numbers)
    high = max(numbers)

    # Tamanho de cada classe
    width = (high - low)/n
    classes = []
    a = low
    b = low + width
    classes = []

    while a < (high-width):
        classes.append((a, b))
        a = b
        b = a + width

    # A última classe pode ser menor que a largura
    classes.append((a, high+1))
    return classes

def classify(numbers, classes):
    # Cria uma lista com o mesmo número de elementos que o número de classes
    count = [0]*len(classes)
    for n in numbers:
        for index, c in enumerate(classes):
            if n >= c[0] and n < c[1]:
                count[index] += 1
                break
    return count

def read_data(filename):
    numbers = []
    with open(filename) as f:
        for line in f:
            numbers.append(float(line))
    return numbers

if __name__ == '__main__':

    num_classes = int(input('Digite o número desejado de classes: '))
    numbers = read_data('arquivos/notas.txt')

    classes = create_classes(numbers, num_classes)
    count = classify(numbers, classes)
    for c, cnt in zip(classes, count):
        print('\n{0:.2f} - {1:.2f} \t {2}'.format(c[0], c[1], cnt))
```

menor número

maior número

nº de classes digitado pelo usuário

```
Digite o número desejado de classes: 4

10.50 - 12.88    4

12.88 - 15.25    5

15.25 - 17.62    7
```

Exercício 9 - Escreva um programa em Python que receba um arquivo com valores e calcule as principais medidas estatísticas: média, mediana, moda, variância e desvio padrão.

Dica: Crie uma função Python para cada medida estatística e uma função para leitura do arquivo. Depois faça chamada a cada função para calcular as estatísticas no arquivo.

In [11]:

```python
# Solução

from collections import Counter

def mean(numbers):
    s = sum(numbers)
    N = len(numbers)
    mean = s/N

    return mean

def median(numbers):
    # Número de itens
    N = len(numbers)

    # Ordenar a lista em ordem crescente
    numbers.sort()

    # Calcula a mediana
    if N % 2 == 0:
        # Se N é par
        m1 = N/2
        m2 = (N/2) + 1
        # Converter para inteiro, corresponder posição
        m1 = int(m1) - 1
        m2 = int(m2) - 1
        median = (numbers[m1] + numbers[m2])/2
    else:
        m = (N+1)/2
        # Converter para inteiro, corresponder posição
        m = int(m) - 1
        median = numbers[m]

    return median

def mode(numbers):
    c = Counter(numbers)
    mode = c.most_common(1)
    return mode[0][0]

def find_differences(numbers):
    m = mean(numbers)
    # Encontrar as diferenças da média
    diff = []
    for num in numbers:
        diff.append(num-m)

    return diff

def variance_sd(numbers):

    # Encontra a lista de diferenças
    diff = find_differences(numbers)

    # Encontra as diferenças ao quadrado
    squared_diff = []
    for d in diff:
        squared_diff.append(d**2)
```

média

Soma dos numeros dividido Pela quantidade de elementos

mediana

Quantos itens

Ne central

1 2 3 5 7 8 10

moda

o Ne que mais aparece

1 2 3 3 4 5

subtrai cada valor Pela média

Vetor com respostas

1 2 3 4 = 10
medio 2,5
1 - 2,5 = -1,5
2 - 2,5 = -0,5
3 - 2,5 = 0,5
4 - 2,5 = 1,5

```python
    # Encontra a variância
    sum_squared_diff = sum(squared_diff)
    variance = sum_squared_diff/len(numbers)
    return variance, variance**0.5

def read_data(filename):
    numbers = []
    with open(filename) as f:
        for line in f:
            numbers.append(float(line))

    return numbers

if __name__=='__main__':
    data = read_data('arquivos/dados.txt')
    m = mean(data)
    median = median(data)
    mode = mode(data)
    variance, sd = variance_sd(data)
    print('Média: {0:.5f}'.format(m))
    print('Mediana: {0:.5f}'.format(median))
    print('Moda: {0:.5f}'.format(mode))
    print('Variância: {0:.5f}'.format(variance))
    print('Desvio Padrão: {0:.5f}'.format(sd))
```

```
Média: 528.58333
Mediana: 500.00000
Moda: 400.00000
Variância: 126683.74306
Desvio Padrão: 355.92660
```

Exercício 10 - Escreva um programa em Python que dados 3 números, resolva a equação quadrática

Nota: Em matemática, uma equação quadrática ou equação do segundo grau é uma equação polinomial de grau dois.

Dica: Você pode usar o módulo cmath para resolver a equação quadrática usando Python. Isso ocorre porque as raízes das equações quadráticas podem ser complexas por natureza.

Equação quadrática: ax**2 + bx + c = 0

In [12]:

```
# Solução

import cmath

a = int(input("Digite o valor de a: "))
b = int(input("Digite o valor de b: "))
c = int(input("Digite o valor de c: "))

# Discriminante
d = (b**2) - (4*a*c)
    ↳ elevado ao quadrado
root1 = (-b - cmath.sqrt(d)) / (2 * a) → Raiz 1
root2 = (-b + cmath.sqrt(d)) / (2 * a) → Raiz 2

print("\n")
print(root1)
print(root2)
```

```
Digite o valor de a: 1
Digite o valor de b: 8
Digite o valor de c: 4


(-7.464101615137754+0j)
(-0.5358983848622456+0j)
```

Parte 2 - Escalares, Vetores, Matrizes e Tensores

In [13]:

```
# NumPy http://www.numpy.org/
import numpy as np
```

Tipos de Dados e Shapes

A maneira mais comum de trabalhar com números em NumPy é através de objetos ndarray. Eles são semelhantes às listas em Python, mas podem ter qualquer número de dimensões. Além disso, o ndarray suporta operações matemáticas rápidas, o que é exatamente o que queremos.

Como você pode armazenar qualquer número de dimensões, você pode usar ndarrays para representar qualquer um dos tipos de dados que abordamos antes: escalares, vetores, matrizes ou tensores.

Scalar

Escalares em NumPy são mais eficientes do que em Python. Em vez dos tipos básicos do Python como int, float, etc., o NumPy permite especificar tipos mais específicos, bem como diferentes tamanhos. Então, em vez de usar int em Python, você tem acesso a tipos como uint8, int8, uint16, int16 e assim por diante, ao usar o NumPy.

Esses tipos são importantes porque todos os objetos que você cria (vetores, matrizes, tensores) acabam por armazenar escalares. E quando você cria uma matriz NumPy, você pode especificar o tipo (mas cada item na matriz deve ter o mesmo tipo). Nesse sentido, os arrays NumPy são mais como arrays C do que as listas em Python.

Exercício 11 - Crie uma matriz NumPy que contenha um escalar, usando a função array do NumPy

In [14]:

```
# Solução

s = np.array(8)
```

Você ainda pode realizar matemática entre ndarrays, escalares NumPy e escalares Python normais, como veremos mais adiante.

Exercício 12 - Qual o shape de um Escalar?

In [15]:

```
# Solução

# Você pode ver o shape da matriz usando o atributo shape, conforme abaixo.
# Esse comando retorna um () vazio, indicando que este objeto é um escalar.

s.shape
```

Out[15]:

```
()
```

é um escalar

Mesmo que os escalares estejam dentro de arrays, você ainda os usa como um escalar normal, para operações matemáticas:

In [16]:

```
x = s - 2
```

Você pode subtrair de um escalar

In [17]:

```
x
```

Out[17]:

```
6
```

E x seria igual a 6. Se você verificar o tipo de x, vai perceber que é numpy.int64, pois você está trabalhando com tipos NumPy, e não com os tipos Python.

Mesmo os tipos escalares suportam a maioria das funções de matriz. Então você pode chamar x.shape e retornaria () porque tem zero dimensões, mesmo que não seja uma matriz. Se você tentar usar o objeto como um escalar Python normal, você obterá um erro.

In [18]:

```
type(x)
```

Out[18]:

```
numpy.int64
```

Vetores

Exercício 13 - Crie um vetor com NumPy a partir de uma lista em Python e visualize o shape

In [19]:

```
# Solução

vec = np.array([1,2,3])
```

Você pode adicionar + de 1 n=

vetor não é uma matriz

In [20]:

```
vec.shape
```

Out[20]:

```
(3,)
```

Apenas uma dimensão

Se você verificar o atributo de shape do vetor, ele retornará um único número representando o comprimento unidimensional do vetor. No exemplo acima, vec.shape retorna (3,).

Agora que há um número, você pode ver que a forma é uma tupla com os tamanhos de cada uma das dimensões do ndarray. Para os escalares, era apenas uma tupla vazia, mas os vetores têm uma dimensão, então a tupla inclui um número e uma vírgula. (Python não entende (3) como uma tupla com um item, por isso requer a vírgula. Documentação oficial do Python 3.6 sobre Tuplas: https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences (https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences)

Você pode acessar um elemento dentro do vetor usando índices, como este abaixo (como você pode ver, em Python a indexaçã começa por 0 e o índice 1 representa o segundo elemento da matriz).

In [21]:

indexação em python começa em zero

```
vec[1]
```

Out[21]:

```
2
```

NumPy também suporta técnicas avançadas de indexação. Por exemplo, para acessar os itens do segundo elemento em diante, você usaria:

In [22]:

```
vec[1:]
```

Out[22]:

```
array([2, 3])
```

NumPy slicing é bastante poderoso, permitindo que você acesse qualquer combinação de itens em um ndarray. Documentação oficial sobre indexação e slicing de arrays: https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html (https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html)

Matrizes

Você cria matrizes usando a função de array() NumPy, exatamente como você fez com os vetores. No entanto, em vez de apenas passar uma lista, você precisa fornecer uma lista de listas, onde cada lista representa uma linha.

Exercício 14 - Crie uma matriz 3x3 contendo os números de um a nove e visualize o shape

In [23]:

```
# Solução

m = np.array([[1,2,3], [4,5,6], [7,8,9]])
```

In [24]:

```
m.shape
```

Out[24]:

```
(3, 3)
```

matriz

Verificando o atributo shape, retornaria a tupla (3, 3) para indicar que a matriz tem duas dimensões, cada dimensão com comprimento 3.

Você pode acessar elementos de matrizes como vetores, mas usando valores de índice adicionais. Então, para encontrar o número 6 na matriz acima, você usaria:

In [25]:

```
m
```

Out[25]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

0 1 2

0
1
2

In [26]: Linha Coluna
1 2

```
m[1][2]
```

Out[26]:

6

Tensores

Os tensores são como vetores e matrizes, mas podem ter mais dimensões.

Exercício 15 - Crie um tensor 3x3x2x1 e visualize o shape

In [27]: Apenas conceito

```
# Solução

t = np.array([[[[1],[2]],[[3],[4]],[[5],[6]]],[[[7],[8]],\
    [[9],[10]],[[11],[12]]],[[[13],[14]],[[15],[16]],[[17],[17]]]])
```

In [28]:

```
t.shape
```

Out[28]: + de 2 dimensões é um tensor

(3, 3, 2, 1)

Exercício 16 - Imprima na tela um único elemento do tensor

Nota: Para acessar um elemento do tensor, usamos a indexação da mesma forma que fazemos com vetores e matrizes

In [29]:

```
t
```

Out[29]:

```
array([[[[ 1],
         [ 2]],

        [[ 3],
         [ 4]],

        [[ 5],
         [ 6]]],


       [[[ 7],
         [ 8]],

        [[ 9],
         [10]],

        [[11],
         [12]]],


       [[[13],
         [14]],

        [[15],
         [16]],

        [[17],
         [17]]]])
```

In [30]:

```
# Solução

t[2][2][1][0]
```

Out[30]:

```
17
```

Alterando o Formato (shape)

Às vezes, você precisará alterar a forma de seus dados sem realmente alterar seu conteúdo. Por exemplo, você pode ter um vetor, que é unidimensional, mas precisa de uma matriz, que é bidimensional. Há duas maneiras pelas quais você pode fazer isso.

Digamos que você tenha o seguinte vetor:

In [31]:

```
vec = np.array([1,2,3,4])
```

In [32]:

```
vec.shape
```

Out[32]:

```
(4,)
```

Exercício 17 - Altere o shape do vetor acima

In [33]:

```
# Solução

x = vec.reshape(1,4)
```

Você consegue alterar o shape transformando um vetor em matriz

In [34]:

```
x.shape
```

Out[34]:

```
(1, 4)
```

Exercício 18 - Suponha que você tenha uma lista de números e que você deseja adicionar 5 a cada item da lista. Sem NumPy, você pode fazer algo como isto:

In [35]:

```
# Lista de valores
valores = [1, 2, 3, 4, 5]

# Looop for para adicionar 5 a cada elemento da lista
for i in range(len(valores)):
    valores[i] += 5

print(valores)
```

```
[6, 7, 8, 9, 10]
```

Mas usando NumPy, como você resolveria este mesmo problema?

In [36]:

```
# Solução

import numpy as np
valores = [1,2,3,4,5]
valores = np.array(valores) + 5
print(valores)
```

Numpy já sabe que quero somar 5 a cada elemento da lista

```
[ 6  7  8  9 10]
```

Exercício 19 - Digamos que você tenha um objeto chamado "valores" e você queira reutilizá-lo, mas primeiro você precisa definir todos os seus valores em zero. Fácil, basta multiplicar por zero e atribuir

o resultado de volta ao objeto, certo? Como você faria isso?

In [37]:

```
# Solução

valores = [1,2,3,4,5]
valores *= 0
print(valores)
```

Se for necessário zerar um vetor ou matriz, é só multiplicar por Ø

```
[]
```

Exercício 20 - Considerando as duas matrizes abaixo, como você faria a soma entre elas?

In [38]:

```
x = np.array([[1,3],[5,7]])
x
```

Out[38]:

```
array([[1, 3],
       [5, 7]])
```

In [39]:

```
y = np.array([[2,4],[6,8]])
y
```

Out[39]:

```
array([[2, 4],
       [6, 8]])
```

Soma de 2 matrizes

$$\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} = \cancel{}$$

$1 + 2 = 3$

$3 + 4 = 7$

$5 + 6 = 11$

$7 + 8 = 15$

$$\begin{bmatrix} 3 & 7 \\ 11 & 15 \end{bmatrix}$$

In [40]:

```
# Solução

x + y
```

Out[40]:

```
array([[ 3,  7],
       [11, 15]])
```

Parte 3 - Operações com Matrizes

Exercício 21 - Dada a matriz abaixo, multiplique 0.5 por cada elemento da matriz

In [41]:

```
import numpy as np
m = np.array([[1,2,3],[4,5,6]])
m
```

Out[41]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [42]:

```
# Solução

n = m * 0.5
n
```

Out[42]:

```
array([[0.5, 1. , 1.5],
       [2. , 2.5, 3. ]])
```

Exercício 22 - Qual a razão da mensagem de erro ao tentar executar a multiplicação das duas matrizes abaixo?

In [43]:

```
a = np.array([[1,2,3,4],[5,6,7,8]])
a
```

Out[43]:

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [44]:

```
b = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
b
```

Out[44]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [45]:

```
# Se as matrizes tiverem shape incompatível, você recebe a mensagem de erro abaixo.
# Veja que A x B não é a mesma coisa que B x A na multiplicação de matrizes.
np.matmul(b, a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-5cbc6b44f7d2> in <module>()
      1 # Se as matrizes tiverem shape incompatível, você recebe a mensagem
 de erro abaixo.
      2 # Veja que A x B não é a mesma coisa que B x A na multiplicação de m
atrizes.
----> 3 np.matmul(b, a)

ValueError: shapes (4,3) and (2,4) not aligned: 3 (dim 1) != 2 (dim 0)
```

In [46]:

```
# Agora sim

np.matmul(a, b)
```

inverte a ordem porque a ordem importa

Out[46]:

```
array([[ 70,  80,  90],
       [158, 184, 210]])
```

Exercício 23 - Obter a transposição de uma matriz é realmente fácil em NumPy. Basta acessar seu atributo T. Há também a função transpose() que retorna a mesma coisa, mas raramente você verá isso usado em qualquer lugar porque digitar T é muito mais fácil. O que representa a transposta de uma matriz?

In [47]:

```
m = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
m
```

Out[47]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [48]:

```
# Solução

# Em matemática, matriz transposta é a matriz que se obtém da troca de linhas por colunas a
m.T
```

Out[48]:

```
array([[ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11],
       [ 4,  8, 12]])
```

Exemplo em Redes Neurais

Digamos que você tenha as seguintes duas matrizes, chamadas inputs e pesos:

In [49]:

```
inputs = np.array([[-0.27,  0.45,  0.64, 0.31]])
inputs
```

Out[49]:

```
array([[-0.27,  0.45,  0.64,  0.31]])
```

In [50]:

```
inputs.shape
```

Out[50]:

```
(1, 4)
```

In [51]:

```
pesos = np.array([[0.02, 0.001, -0.03, 0.036], \
    [0.04, -0.003, 0.025, 0.009], [0.012, -0.045, 0.28, -0.067]])

pesos
```

Out[51]:

```
array([[ 0.02 ,  0.001, -0.03 ,  0.036],
       [ 0.04 , -0.003,  0.025,  0.009],
       [ 0.012, -0.045,  0.28 , -0.067]])
```

In [52]:

```
pesos.shape
```

Out[52]:

```
(3, 4)
```

Os pesos são multiplicados aos inputs nas camadas das redes neurais, logo precisamos multiplicar pesos por inputs. Uma multiplicação de matrizes. Isso é muito fácil com o NumPy.

In [53]:

```
np.matmul(inputs, pesos)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-d190cbc25737> in <module>()
----> 1 np.matmul(inputs, pesos)

ValueError: shapes (1,4) and (3,4) not aligned: 4 (dim 1) != 3 (dim 0)
```

Exercício 24 - Ops, o que deu errado? Se você fez o exercício de multiplicação de matrizes, então você já viu esse erro antes. As matrizes estão com shapes incompatíveis porque o número de colunas na matriz esquerda, 4, não é igual ao número de linhas na matriz direita, 3.

Como resolvemos isso? Podemos usar a transposta da matriz:

In [54]:

```
np.matmul(inputs, pesos.T)
```

converte as duas matrizes em linhas

Out[54]:

```
array([[-0.01299,  0.00664,  0.13494]])
```

Também funciona se você gerar a transposta da matriz de inputs e trocar a ordem dos parâmetros na função:

In [55]:

```
np.matmul(pesos, inputs.T)
```

Out[55]:

```
array([[-0.01299],
       [ 0.00664],
       [ 0.13494]])
```

converte as duas matrizes em linhas.

As duas respostas são transpostas uma da outra, de modo que a multiplicação que você usa realmente depende apenas da forma (shape) que deseja para a saída.

Desafio

Use as duas matrizes abaixo para resolver os exercícios de 25 a 28

$$\mathbf{A} = \begin{bmatrix} 3 & 2 & -1 \\ 6 & 4 & -2 \\ 5 & 0 & 3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 3 & 2 \\ 3 & -4 & -2 \\ 4 & -1 & 1 \end{bmatrix}$$

Exercício 25 - Qual o Rank da Matriz A?

Na álgebra linear, o Rank de uma matriz A é a dimensão do espaço vetorial gerado (ou estendido) por suas colunas. Isso corresponde ao número máximo de colunas linearmente independentes de A. Isso, por sua vez, é idêntico à dimensão do espaço ocupado por suas linhas. Rank é, portanto, uma medida da "não-degeneração" do sistema de equações lineares e transformação linear codificada por A. Existem várias definições equivalentes de Rank e o Rank de uma matriz é uma de suas características mais fundamentais.

In [56]:

```
# Solução

import numpy as np
import numpy.linalg as linalg

# Criando as matrizes em formato NumPy
A = np.array([ [3, 2, -1], [6, 4, -2], [5, 0, 3]])
B = np.array([ [2, 3, 2], [3, -4, -2], [4, -1, 1]])

# Calculamos o Rank da matriz A
linalg.matrix_rank(A)
```

Out[56]:

```
2
```

Exercício 26 - Calcule $\mathbf{A} \cdot \mathbf{B}$

In [57]:

```
# Solução

A @ B
```

Só funciona no jupyter
mesmo coisa que o função dot

Out[57]:

```
array([[ 8,  2,  1],
       [16,  4,  2],
       [22, 12, 13]])
```

In [58]:

```
np.dot(A, B)
```

Out[58]:

```
array([[ 8,  2,  1],
       [16,  4,  2],
       [22, 12, 13]])
```

Exercício 27 - Qual o segundo autovetor em $\mathbf{B}$?

In [59]:

```
# Solução

eig_vals, eig_vecs = linalg.eig(B)
eig_vecs[:,1]
```

Retorna auto valores e auto vetores

Out[59]:

```
array([-0.66610619, -0.0601505 , -0.74342751])
```

Exercício 28 - Resolva $\mathbf{B}\vec{x} = \vec{b}$ onde $\vec{b} = [14, -1, 11]$

In [60]:

```
# Solução

b = np.array([14, -1, 11])
linalg.solve(B, b)
```

Out[60]:

```
array([3., 2., 1.])
```

Exercício 29 - Calcule a inversa da matriz abaixo

Uma matriz quadrada A é dita invertível quando existe outra matriz denotada A^-1 tal que

A^-1 . A = I

e

A . A^-1 = I

A inversa de uma matriz pode ser encontrada usando o comando `linalg.inverse` . Considere o seguinte sistema de equações:

$$\begin{aligned} 3x + 2y + z &= 5 \\ 2x - y &= 4 \\ x + y - 2z &= 12 \end{aligned}$$

Podemos codificá-lo como uma equação matricial:

x

$$\begin{bmatrix} 3 & 2 & 1 \\ 2 & -1 & 0 \\ 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 12 \end{bmatrix}$$

matriz A

matriz B

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A}^{-1}\mathbf{b} = \mathbf{x}$$

inversa da matriz A vezes b igual a x

Crie o código Python para resolver a equação matricial acima.

In [61]:

\ apenas p/ quebrar linha

```
# Criando as matrizes em Python
a_matrix = [[3, 2, 1],\
            [2,-1,0],\
            [1,1,-2]]

b_matrix = [5, 4, 12]

# Convertendo as matrizes para o formato NumPy
np_a_matrix = np.array(a_matrix)
np_b_matrix = np.array(b_matrix).transpose()

# Resolvendo o problema
np_a_inv = linalg.inv(np_a_matrix)
np_x_matrix = np_a_inv.dot(np_b_matrix)

# Print
print(np_x_matrix)
```

encontramos a inversa da matriz A

multiplicação

```
[ 2.47058824  0.94117647 -4.29411765]
```

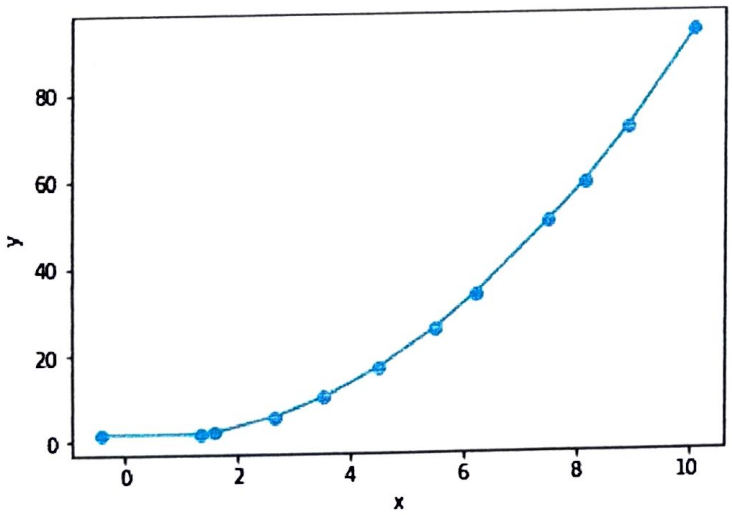
Exercício 30 - Faça um Plot, usando Matplotlib, que mostre a relação entre x e y

In [62]:

```
# Solução

x = [-0.42,1.34,1.6,2.65,3.53,4.48,5.48,6.21,7.49,8.14,8.91,10.1]
y = [1.58,1.61,2.04,5.47,9.8,16.46,25.34,33.32,49.7,58.79,71.26,93.34]

x = np.array(x)
y = np.array(y)

import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(x,y, 'o-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

listas em Python

convertidos para vetores



módulo 2

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 2

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Resolvendo Sistemas de Equações em Python

Usando numpy, podemos resolver sistemas de equações em Python. Cada equação em um sistema pode ser representada com matrizes. Por exemplo, se tivermos a equação 3x - 9y = -42, ela pode ser representada como [3, -9] e [-42]. Se adicionarmos outra equação, por exemplo, 2x + 4y = 2, podemos mesclar com a equação anterior para obter [[3, -9], [2, 4]] e [-42, 2]. Agora vamos resolver os valores x e y.

Primeiro, colocamos essas equações em matrizes numpy:

In [1]:

```
import numpy as np

A = np.array([ [3,-9], [2,4] ])
b = np.array([-42,2])
```

Depois, usamos a função linalg.solve() para resolver os valores x e y, assim:

In [2]:

```
x, y = np.linalg.solve(A,b)
print("Os valores de x e y nas equações acima são, respectivamente:", x, y)
```

```
Os valores de x e y nas equações acima são, respectivamente: -5.0 3.0
```

Exercício 1 - Resolva o sistema de equações:

1x + 1y = 35

2x + 4y = 94

In [3]:

```
# Solução

expressao1 = np.array([[1, 1],[2,4]])
expressao2 = np.array([35, 94])

x, y = np.linalg.solve(expressao1, expressao2)
print("Os valores de x e y nas equações acima são, respectivamente:", x, y)
```

```
Os valores de x e y nas equações acima são, respectivamente: 23.0 12.0
```

Potência

Exercício 2 - Resolva a equação quadrática ax**2 + bx + c = 0 usando a fórmula matemática

Dica 1: Solicite ao usuário que digite os valores dos 3 coeficientes da equação (a, b e c) e resolva os valores de x.

Dica 2: Use a função sqrt() do pacote math

Dica 3: Fique atento aos possíveis retornos da equação:

- Se d < 0: Sem solução em $\mathbb{R}$
- Se d = 0: A equação tem apenas uma solução em $\mathbb{R}$, x = −b2a
- Se d > 0: A equação tem duas soluções, x1 e x2

In [4]:

Equação do Segundo Grau

```
# Solução

import math

print('Vou resolver a equação quadrática ax^2 + bx + c = 0. Digite os valores de a, b e c:\

a = int(input('a = '))
b = int(input('b = '))
c = int(input('c = '))

d = b**2-4*a*c

if d < 0:
    print('A equação não tem solução real.')
elif d == 0:
    x = (-b)/(2*a)
    print('A equação tem uma solução: ',x)
else:
    x1 = (-b+math.sqrt(d))/(2*a)
    x2 = (-b-math.sqrt(d))/(2*a)
    print('A equação tem duas soluções: ',x1, 'or', x2)
```

Fórmula de Bhaskara

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2 \cdot a}$$

```
Vou resolver a equação quadrática ax^2 + bx + c = 0. Digite os valores de a,
b e c:

a = 1
b = 9
c = 8
A equação tem duas soluções:  -1.0 or -8.0
```

Exercício 3 - Resolva a equação quadrática ax2 + bx + c = 0, criando uma função Python que receba os 3 coeficientes como parâmetro**

In [5]:

todo o código em uma única linha

```
# Solução

import math

def solvequadratic(a,b,c):
    return(((-1*b)+math.sqrt(b**2-4*a*c))/2*a),(((-1*b)-math.sqrt((b**2)-4*(a)*(c)))/2*a)

solvequadratic(1, 9, 8)
```

-1.0 -8.0

Out[5]:

```
(-1.0, -8.0)
```

Exercício 4 - Estude as funções matemáticas em Python no link:

https://docs.python.org/3.6/library/math.html (https://docs.python.org/3.6/library/math.html)

Exercício 5 - Crie uma classe em Python para calcular um número elevado ao

quadrado

Dica: Para testar, instancie a classe em um objeto, passando o número como parâmetro

In [6]:

```
# Solução

class MathFunction(object):
    def __init__(self, code):
        self.code = code
        self._func = eval("lambda x: " + code)
    def __call__(self, arg):
        return self._func(arg)
    def __repr__(self):
        return "f(x) = " + self.code
```

constructor

converte texto string para código ou número

In [7]:

```
sq = MathFunction("x**2")
```

cria objeto sq recebendo x**2

In [8]:

```
sq
```

Out[8]:

```
f(x) = x**2
```

In [9]:

```
sq(7)
```

chama objeto sq passando 7

code = 7

func: lambda x = 7 : 7^2

x = 49

Out[9]:

```
49
```

Exercício 6 - Crie uma função em Python para resolver a equação quadrática ax2 + bx + c = 0, aplicando tratamento de erros**

sq = MathFunction ("x/2")

sq

f(x) = x/2

sq(7)

3.5

In [10]:

```
# Solução

from numpy import linalg
from numpy import array
from math import sqrt

class NoRootsError(Exception):
    pass
class NoRationalNumbersError(Exception):
    pass
class NotQuadraticError(Exception):
    pass

def returnX(a,b,c):
  if a==0:
    raise NotQuadraticError
  if b**2-4*a*c < 0:
    raise NoRootsError
  try:
    first_solution = (-b + sqrt(b**2-4*a*c))/2*a
    second_solution = (-b - sqrt(b**2-4*a*c))/2*a
    return (first_solution,second_solution)
  except ValueError:
    raise NoRationalNumbersError
```

In [11]:

```
returnX(1, 9, 8)
```

Out[11]:

```
(-1.0, -8.0)
```

Exercício 7 - Crie um Plot com a linha que representa a função linear

Dica 1: Função linear y = mx + b / y = ax + b

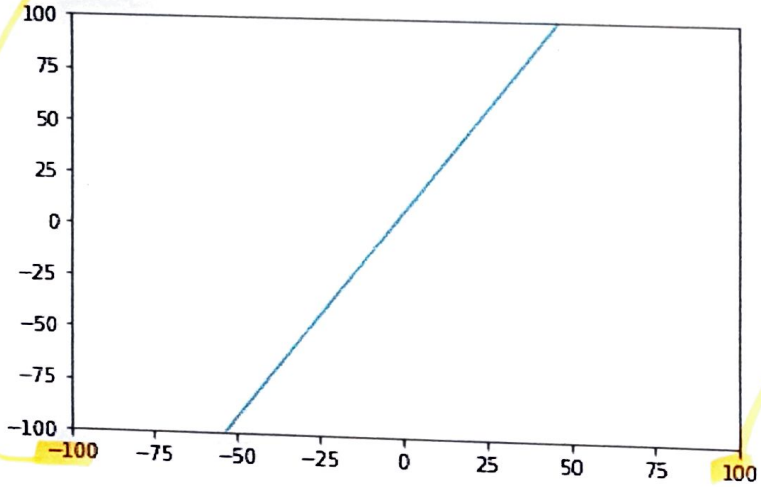
In [12]:

```
# Solução

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

A1 = 2
B2 = 7

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

fig,ax = plt.subplots()

ax.plot(x, A1 * x + B2)


ax.set_xlim((-100.,100.))
ax.set_ylim((-100.,100.))

plt.show()
```

Gero dados

$y = mx + b$

Não importa o letra o que importa é a equação $m * x + b$



Exercício 8 - Crie um Plot com a Parabola que representa a função quadrática

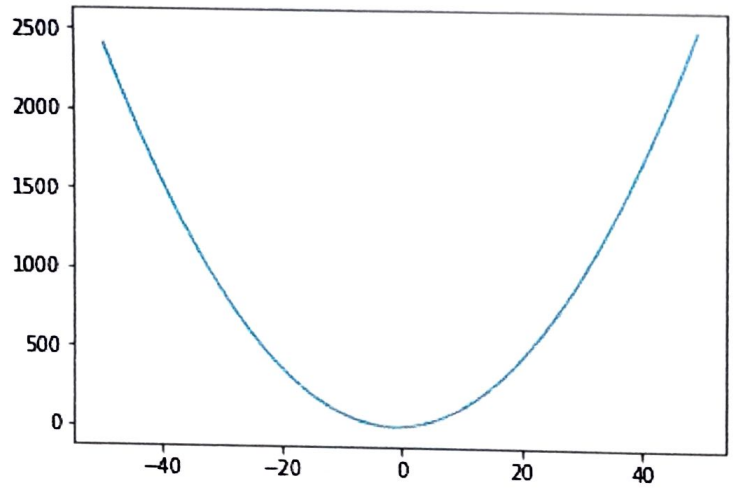
In [13]:

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

a=[]
b=[]

for x in range(-50,50,1):
    y = x ** 2 + 2 * x + 2
    a.append(x)
    b.append(y)

fig= plt.figure()
axes=fig.add_subplot(111)
axes.plot(a,b)
plt.show()
```

→ duas listas vazias

→ cria um range de valores

função quadrática



Desafio - Questões 9 e 10

Exercício 9 - Crie uma função que calcule o slope e o intercepto (coeficientes m e b) da função linear estudada nas aulas anteriores, calcule e desenhe a linha que representa a função

Dica 1: Use dados dummy para x e y

Dica 2: A fórmula da função linear é y = mx+b

In [14]:

```
# Solução

from statistics import mean
import numpy as np

xs = np.array([1,2,3,4,5], dtype=np.float64)
ys = np.array([5,4,6,5,6], dtype=np.float64)

def best_fit_slope_and_intercept(xs,ys):
    m = (((mean(xs)*mean(ys)) - mean(xs*ys)) / ((mean(xs)*mean(xs)) - mean(xs*xs)))

    b = mean(ys) - m*mean(xs)

    return m, b

m, b = best_fit_slope_and_intercept(xs,ys)

print(round(m, 2), round(b,2))
```

vetores tipo float

```
0.3 4.3
```

list Comprehension

In [15]:

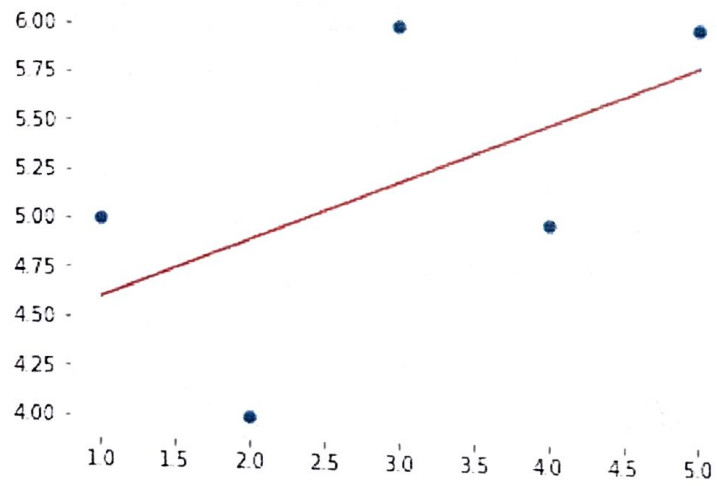
```
# Calcula a linha usando list comprehension
line = [(m*x)+b for x in xs]
```

[x+1 for x in [1,2,3,4,5]]

[2,3,4,5,6]

In [16]:

```
# Plot
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')

plt.scatter(xs,ys,color='#003F72')
plt.plot(xs, line)
plt.show()
```



Exercício 10 - Defina um novo valor de x e encontre o y correspondente

In [17]:

```
# Solução

# Novo valor de x
predict_x = 7
```

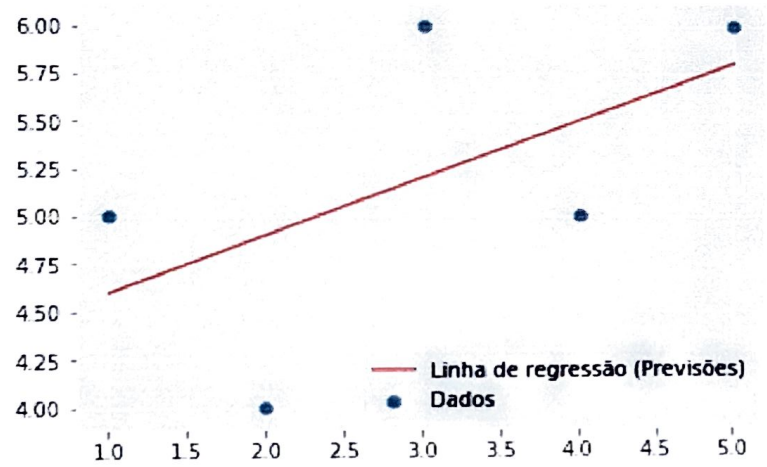
In [18]:

```
# Prevendo o valor de y
predict_y = (m*predict_x)+b
print(predict_y)
```

```
6.399999999999996
```

In [19]:

```
# Plot
predict_x = 7
predict_y = (m*predict_x)+b

plt.scatter(xs,ys,color='#003F72',label='Dados')
plt.plot(xs, line, label='Linha de regressão (Previsões)')
plt.legend(loc=4)
plt.show()
```



Fim

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 3 (Continuação Fundamentos da Matemática)

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

A maioria das pessoas considera a Matemática e a Física, assustadoras bestas das quais é melhor manter a distância. Computadores, no entanto, podem nos ajudar a domar a complexidade e a aritmética tediosa das manipulações associadas a esses assuntos. De fato, Matemática e Física são muito mais acessíveis, uma vez que você tem o poder de computadores do seu lado.

Divirta-se!!

Exercício 1 - Crie uma função em Python que retorne o resultado do polinômio abaixo. Em seguida, defina x = 3 e execute a função.

In [1]:

```
from IPython.display import Image
Image("images/img1.png")
```

Out[1]:

$$p(x) = x^4 - 4 \cdot x^2 + 3 \cdot x$$

Dica: Represente o polinômio usando operações aritméticas no retorno da sua função. Para x = 3 o resultado deve ser 54.

In [2]:

```
# Solução
def p(x):
    return x**4 - 4*x**2 + 3*x

x = 3
p(x)
```

Out[2]:

```
54
```

Exercício 2 - Dado o polinômio ax^3 + bx^2 + cx + d, solicite que o usuário digite o valor dos 4 coeficientes e o valor de x e resolva o polinômio.

Dica 1: Use a função pow() do pacote math para representar operações de potência.

Dica 2: Armazene os coeficientes digitados em uma lista Python.

In [3]:

```
# Solução

import math
print("Digite os coeficientes da fórmula ax^3 + bx^2 + cx + d")

# Inicializando a lista
lst = []  inicializa uma lista vazia

# Lendo os coeficientes e armazenando na lista
for i in range(0,4):  4 não conta
    a = int(input("Digite o coeficiente: "))
    lst.append(a)

x = int(input("Dogite o valor de x: "))

sum1 = 0

j = 3

for i in range(0,3):  executa 3 vezes
                      0 -> 2
    while(j > 0):
        sum1 = sum1 + (lst[i] * math.pow(x,j))
        break
                      ABC
    j = j - 1

sum1 = sum1 + lst[3]
              D

print("O valor do polinômio é:", sum1)
```

insere 4 itens

[A, B, C, D]

0, 1, 2, 3

x^3

x^2

x^1

```
Digite os coeficientes da fórmula ax^3 + bx^2 + cx + d
Digite o coeficiente: 2
Digite o coeficiente: 3
Digite o coeficiente: 5
Digite o coeficiente: 1
Dogite o valor de x: 4
O valor do polinômio é: 197.0
```

Matemática com Sympy ⇒ colocado no p/ ~~python~~ python

O Sympy é um excelente pacote matemático para operações mais avançadas e facilita nosso trabalho em várias situações. Além disso, ele permite formatar as fórmulas de modo que a visualização fique mais amigável e profissional. Para isso, use init_printing() depois de importar o Sympy assim:

In [4]:

```
#!pip install sympy
```

In [5]:

importa tudo

```
from sympy import *
init_printing()
```

Exercício 3 - Usando o Sympy, resolva a equação $x^2 + 2x - 8 = 0$

In [6]:

```
# Solução
x = symbols('x')
solve( x**2 + 2*x - 8, x)
```

Out[6]:

$[-4, \quad 2]$

Exercício 4 - Use Sympy para apresentar a equação $ax^2 + bx + c = 0$ em seu formato simbólico

In [7]:

Se não usar o Symbols vira dar a resposta

```
# Solução
x = symbols('x')
a, b, c = symbols('a b c')
solve( a*x**2 + b*x + c, x)
```

Out[7]:

Representação do fórmula

$$\left[\frac{1}{2a}\left(-b + \sqrt{-4ac + b^2}\right), \quad -\frac{1}{2a}\left(b + \sqrt{-4ac + b^2}\right)\right]$$

Exercício 5 - Use o "subs" do Sympy, para plugar os valores 1, 2 e -8 para os coeficientes a, b e c, respectivamente, na equação do Exercício 4.

Dica: A função subs() recebe um dicionário Python como entrada, com a chave sendo o coeficiente e o valor, sendo o valor numérico.

{} dicionário
[] lista
() tuplo

In [8]:

```
# Solução
a, b, c = symbols('a b c')
gen_sol = solve( a*x**2 + b*x + c, x)
[ gen_sol[0].subs({'a':1,'b':2,'c':-8}),
  gen_sol[1].subs({'a':1,'b':2,'c':-8}) ]
```

Out[8]:

$[2, \quad -4]$ 2 soluções possíveis pois é uma equação quadrática

Um Pouco mais sobre o Sympy

Para resolver um sistema *de equações* , você pode alimentar a função `solve` com a lista de equações como o primeiro argumento, e especifique a lista de incógnitas que você quer resolver como o segundo argumento. Por exemplo, para resolver por x e y no sistema de equações $x + y = 3$ e $3x - 2y = 0$, use:

In [9]:

```
x, y = symbols('x y')
solve([x + y - 3, 3*x - 2*y], [x, y])
```

Out[9]:

$$\left\{x: \frac{6}{5},\quad y: \frac{9}{5}\right\}$$

A função `solve` é como um canivete suíço que você pode usar para resolver todo tipo de problema. Suponha que você queira *completar o quadrado* no expressão $x^2 - 4x + 7$, ou seja, você quer encontrar constantes h e k tal que $x^2 - 4x + 7 = (x - h)^2 + k$. Não há uma função especial "completar o quadrado " em `SymPy` , mas você pode resolver a equação $(x - h)^2 + k - (x^2 - 4x + 7) = 0$para encontrar as incógnitas h e k:

In [10]:

```
h, k = symbols('h k')
solve( (x-h)**2 + k - (x**2-4*x+7), [h,k] )
```

Out[10]:

$$[(2,\quad 3)]$$

Aprenda os comandos básicos do `SymPy` e você nunca precisará sofrer outro cálculo aritmético entediante meticulosamente realizado à mão novamente! :-)

Exercício 6 - Usando a função expand() do Sympy, encontre a versão estendida do Polinômio P, abaixo.

Dica: A versão estendida do polinômio é simplesmente aplicar as operações algébricas ao polinômio

In [11]:

```
P = (x-1)*(x-2)*(x-3)
```

In [12]:

```
# Solução
P.expand()
```

Out[12]:

$$x^3 - 6x^2 + 11x - 6$$

Quando o polinômio é expresso em sua forma expandida $P(x) = x^3 - 6x^2 + 11x - 6$, não podemos identificar imediatamente suas raízes. Esta é a razão pela qual a forma fatorada $P(x) = (x - 1)(x - 2)(x - 3)$ é preferível. Para fatorar um polinômio, chame o método `factor` ou `simplify` :

In [13]:

```
P.factor()
```

Out[13]:

$(x-3)(x-2)(x-1)$

Volta a equação para sua forma de origem

In [14]:

```
P.simplify()
```

Out[14]:

$(x-3)(x-2)(x-1)$

Lembre-se de que as raízes do polinômio $P(x)$ são definidas como soluções para a equação $P(x) = 0$. Nós podemos usar a função `solve` para encontrar as raízes do polinômio:

In [15]:

```
roots = solve(P,x)
roots
```

Out[15]:

$[1, \quad 2, \quad 3]$

In [16]:

```
# Vamos checar se P é igual a (x-1) (x-2) (x-3)
simplify( P - (x-roots[0])*(x-roots[1])*(x-roots[2]) )
```

Verifico se as raízes são iguais a zero

Out[16]:

? ?

0

Números Complexos

Desde Newton, a palavra "número" tem sido usada para se referir a um dos seguintes tipos de objetos matemáticos:

Os naturais $\mathbb{N}$, os inteiros $\mathbb{Z}$, os racionais $\mathbb{Q}$ e os números reais $\mathbb{R}$.

Cada conjunto de números é associado a uma classe diferente de equações.

Os números naturais $\mathbb{N}$ aparecem como soluções da equação $m + n = x$, onde m e n são números naturais (denotados $m, n \in \mathbb{N}$).

Os inteiros $\mathbb{Z}$ são as soluções às equações da forma $x + m = n$, onde $m, n \in \mathbb{N}$.

Os racionais $\mathbb{Q}$ são necessários para resolver x em $mx = n$, com $m, n \in \mathbb{Z}$.

As soluções para $x^2 = 2$ são irracionais (então $\notin \mathbb{Q}$) então precisamos de um conjunto maior que contém *todos* os números possíveis.

Um padrão surge onde equações mais complicadas exigem invenção de novos tipos de números.

Considere a equação quadrática $x^2 = -1$. Não há soluções reais a esta equação, mas podemos definir um número imaginário $i = sqrt{-1}$ (denotado `I` in `SymPy`) que satisfaz esta equação:

In [17]:

```
I*I
```

$i / i = 1$

$i + i = 2i$

$i - i = \phi$

Out[17]:

-1

In [18]:

```
x = symbols('x')
solve( x**2 + 1 , x)
```

Out[18]:

$[-i, \quad i]$

As soluções são $x = i$ e $x = -i$ e, de fato, podemos verificar que $i^2 + 1 = 0$ e $(-i)^2 + 1 = 0$ desde $i^2 = -1$.

Os números complexos $\mathbb{C}$ são definidos como $\{a + bi \mid a, b \in \mathbb{R}\}$. Números complexos contém uma parte real e uma parte imaginária:

In [19]:

```
z = 4 + 3*I
z
```

Out[19]:

$4 + 3i$

Exercício 7 - Pesquise na documentação do Sympy as funções necessárias para extrair o módulo (valor absoluto) de z e seu argumento.

In [20]:

```
# Solução
```

In [21]:

```
# Módulo
Abs(z)
```

calculo valor absoluto

Out[21]:

5

In [22]:

```
# Argumento
arg(z)
```

Out[22]:

$$\operatorname{atan}\left(\frac{3}{4}\right)$$

Exercício 8 - Desafio e Pesquisa

[A fórmula de Euler] (https://en.wikipedia.org/wiki/Euler's_formula (https://en.wikipedia.org/wiki/Euler's_formula)) mostra uma relação importante entre a função exponencial e^x e as funções trigonométricas $sin(x)$ e $cos(x)$:

$$e^{ix} = \cos x + i \sin x.$$

Para obter este resultado em `SymPy`, você deve especificar que o número x é real e também `expandir` a equação considerando números complexos.

Pesquise sobre a fórmula de Euler e implemente a fórmula acima com Sympy. Algumas distribuições de máxima entropia que usamos em Machine Learning são baseadas na fórmula de Euler.

In [23]:

```
# Solução
x = symbols('x', real=True)
exp(I*x).expand(complex=True)
```

x pertence ao conjunto de Nº reais

expand p/ ver com Nº complexo

exponencial

Out[23]:

$$i \sin(x) + \cos(x)$$

Exercício 9 - Resolva o sistema de equações lineares abaixo usando a função solve() do Sympy.

x + y + z = 1

x + y + 2z = 3

In [24]:

```
# Solução
x, y, z = symbols('x, y, z')
eq1 = x + y + z
eq2 = x + y + 2*z
solve([eq1-1, eq2-3], (x, y,z))
```

definindo Simbolos

Valor da equação 1

Valor da equação 2

Devolva o valor de x y z

resolva equação 1 onde 1 é o resultado

Resolva a equação 2 onde 3 é o resultado

Out[24]:

$$\{x : -y - 1,\ z : 2\}$$

Exercício 10 - Encontre o logaritmo de 1.5 na base 2, base 10 e o logaritmo

natural

Dica: O pacote math oferece funções para as 3 operações.

In [25]:

```
# Solução
import math
```

In [26]:

```
# Base 2
print( math.log2(1.5))
```

calcula log no base 2

```
0.5849625007211562
```

In [27]:

```
# Base 2
print( math.log(1.5, 2))
```

calcula log natural No base 2

```
0.5849625007211562
```

In [28]:

```
# Base 10
print( math.log10(1.5))
```

Calculo log no base 10

```
0.17609125905568124
```

In [29]:

```
# Logaritmo natural (base e)
print( math.log(1.5))
```

log no base natural

```
0.4054651081081644
```

Fim

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 4

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

A maioria das pessoas considera a Matemática e a Física, assustadoras bestas das quais é melhor manter a distância. Computadores, no entanto, podem nos ajudar a domar a complexidade e a aritmética tediosa das manipulações associadas a esses assuntos. De fato, Matemática e Física são muito mais acessíveis, uma vez que você tem o poder de computadores do seu lado.

Divirta-se!!

Exercício 1 - Considerando os vetores v [1, 2] e w [2, 1], crie um script Python que faça a soma dos vetores. Não use funções de pacotes prontos, somente a linguagem Python pura e suas funções, se necessário.

Dica: A função zip() pode ajudá-lo.

List comprehension

In [1]:

```
# Solução

vetor1 = [1, 2]
vetor2 = [2, 1]

def soma_vetor(v, w):
    soma = [v_i + w_i for v_i, w_i in zip(v, w)]
    return soma

soma_vetor(vetor1, vetor2)
```

zip pega as informações nas variáveis v e w e joga para v_i e w_i

```
x = [1, 2, 3]
y = [4, 5, 6]
for t in zip(x, y):
    print(t)
```

(1, 4)
(2, 5)
(3, 6)

List Comprehension trabalha com os valores recebidos pelo for loop um a um enviando para a variável em questão

Out[1]:

```
[3, 3]
```

Exercício 2 - Considerando os vetores v [1, 2] e w [2, 1], crie um script Python que faça a subtração dos vetores. Não use funções de pacotes prontos,

somente a linguagem Python pura e suas funções, se necessário.

Dica: A função zip() pode ajudá-lo.

In [2]:

```
# Solução

vetor1 = [1, 2]
vetor2 = [2, 1]

def subtrai_vetor(v, w):
    sub = [v_i - w_i for v_i, w_i in zip(v, w)]
    return sub

subtrai_vetor(vetor1, vetor2)
```

Out[2]:

```
[-1, 1]
```

Exercício 3 - Considerando o escalar e vetor abaixo, crie uma função que receba os 2 como parâmetro e multiplique o vetor pelo escalar. Use apenas Python puro.

Dica: List Comprehension pode ajudá-lo aqui. O curso de Python Fundamentos oferece exemplos deste recurso.

In [3]:

```
# Solução

num = 3
vetor = [1, 2, 3]

def scalar_multiply(c, v):
    resultado = [c * v_i for v_i in v]
    return resultado

scalar_multiply(num, vetor)
```

escalar

vetor

[1,2,3]

list Comprehension

Out[3]:

```
[3, 6, 9]
```

3x1 = 3
3x2 = 6
3x3 = 9

Exercício 4 - Considerando os 2 vetores abaixo, crie uma função Python que faça o dot product entre os 2 vetores e retorne apenas um valor escalar.

In [4]:

```
# Solução

vetor1 = [1, 2]
vetor2 = [2, 1]

def dot(v, w):
    resultado = sum(v_i * w_i for v_i, w_i in zip(v, w))
    return resultado

dot(vetor1, vetor2)
```

Apenas o nome do função, não é o nome usado

list comprehension dentro do função sum

depois de multiplicar os elementos do listo ele soma

Out[4]:

4

$1 \times 2 = 2$

$2 \times 1 = 2$

$2 + 2 = \boxed{4}$

Exercício 5 - Resolva o exercício anterior usando a função dot() do NumPy

In [5]:

```
# Solução

import numpy as np

vetor1 = [1, 2]
vetor2 = [2, 1]

np.dot(vetor1, vetor2)
```

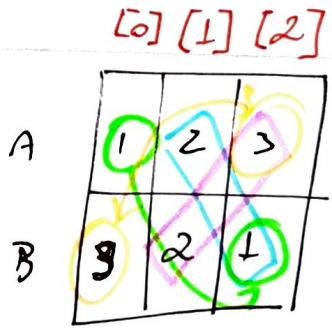
Out[5]:

4

Exercício 5.1 - Considerado os vetores abaixo, encontre o cross product entre eles, usando Python puro e NumPy.

In [6]:

```
# Solução

vetor1 = [1, 2, 3]
vetor2 = [3, 2, 1]

def my_cross(a, b):
    c = [a[1]*b[2] - a[2]*b[1],
         a[2]*b[0] - a[0]*b[2],
         a[0]*b[1] - a[1]*b[0]]

    return c

my_cross(vetor1, vetor2)
```



$2 \times 1 = 2 - 2 \times 3 = 6$

$= -4$

$2 \times 3 = 9 - 1 \times 1 = 1$

$= 8$

$1 \times 2 = 2 - 2 \times 3 = 6$

$= -4$

Out[6]:

[-4, 8, -4]

In [7]:

```
# Solução

import numpy as np

vetor1 = [1, 2, 3]
vetor2 = [3, 2, 1]

np.cross(vetor1, vetor2)
```

Cross é uma função Numpy

Out[7]:

```
array([-4,  8, -4])
```

Exercício 6 - Crie uma lista em Python com 3 dimensões, preenchida com zeros.

Dica: a função pprint do pacote pprint permite imprimir uma lista tridimensional de forma mais legível.

In [8]:

```
# Solução

import pprint
n = 3
vetor = [[[0 for k in range(n)] for j in range(n)] for i in range(n)]
type(vetor)
pprint.pprint(vetor)
```

```
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]
```

Função PPrint. PPrint() imprime os três vetores um embaixo do outro

Exercício 7 - Considerando o vetor criado no item anterior, altere o valor na posição nas coordenadas (2, 1, 0) para 92.

In [9]:

```
# Solução
vetor[2][1][0] = 92
pprint.pprint(vetor)
```

linha
Bloco
coluna

coluna 0 coluna 1 coluna 2

```
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [92, 0, 0], [0, 0, 0]]]
```

linha 0
linha 1
linha 2

Bloco 0 Bloco 1 Bloco 2

Exercício 8 - Considere o vetor multidimensional A abaixo. Transforme-o em um vetor de uma única dimensão, com todos os elementos ordenados na mesma sequência do vetor original.

Dica: Esta operação é chamada de flatten do vetor (algo como "achatar" um vetor), sendo uma das operações de pré-processamento mais comum em modelos de Deep Learning para Visão Computacional. NumPy oferece uma função para isso.

In [10]:

```
# Solução

import numpy as np

A = np.array([[[ 0,  1],
               [ 2,  3],
               [ 4,  5],
               [ 6,  7]],
              [[ 8,  9],
               [10, 11],
               [12, 13],
               [14, 15]],
              [[16, 17],
               [18, 19],
               [20, 21],
               [22, 23]]])

print("\nVetor multidimensional original:\n\n", A)

Flattened_A = A.flatten()
print("\n\nVetor flattened:\n\n", Flattened_A)
```

```
Vetor multidimensional original:

 [[[ 0  1]
  [ 2  3]
  [ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]
  [12 13]
  [14 15]]

 [[16 17]
  [18 19]
  [20 21]
  [22 23]]]


Vetor flattened:

 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

Achatar

uma única dimensão

Exercício 9 - Considere o vetor multidimensional A abaixo. Transforme-o em um vetor de uma única dimensão, com todos os elementos ordenados na mesma sequência do vetor original. Use a função ravel() do NumPy.

A função ravel() possui o parâmetro "order" que pode ser "C", "F", "A" ou "K". Qual a diferença entre usar os parâmetros C e F?

In [11]:

```python
# Solução

import numpy as np

A = np.array([[[ 0,  1],
               [ 2,  3],
               [ 4,  5],
               [ 6,  7]],
              [[ 8,  9],
               [10, 11],
               [12, 13],
               [14, 15]],
              [[16, 17],
               [18, 19],
               [20, 21],
               [22, 23]]])

print("\nVetor multidimensional original:\n\n", A)

Flattened_A = A.ravel(order="C")
print("\n\nVetor flattened com Parâmetro C:\n\n", Flattened_A)
Flattened_A = A.ravel(order="F")
print("\n\nVetor flattened com Parâmetro F:\n\n", Flattened_A)
```

```
Vetor multidimensional original:

 [[[ 0  1]
  [ 2  3]
  [ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]
  [12 13]
  [14 15]]

 [[16 17]
  [18 19]
  [20 21]
  [22 23]]]


Vetor flattened com Parâmetro C:

 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]


Vetor flattened com Parâmetro F:

 [ 0  8 16  2 10 18  4 12 20  6 14 22  1  9 17  3 11 19  5 13 21  7 15 23]
```

Sequencia

muda a sequencia

Exercício 10 - Calcule o Inner e Outer product dos vetores abaixo.

dot Product

inverso do dot

In [12]:

```python
# Solução

import numpy as np

vetor1 = [1, 2]
vetor2 = [2, 1]

innerProduct = np.inner(vetor1, vetor2)
outerProduct = np.outer(vetor1, vetor2)

print(innerProduct)
print(outerProduct)
```

dot Product

```
4
[[2 1]
 [4 2]]
```

[1, 2]
[2, 1]

1 x 2 = 2
1 x 1 = 1

2 x 2 = 4
2 x 1 = 2

outer

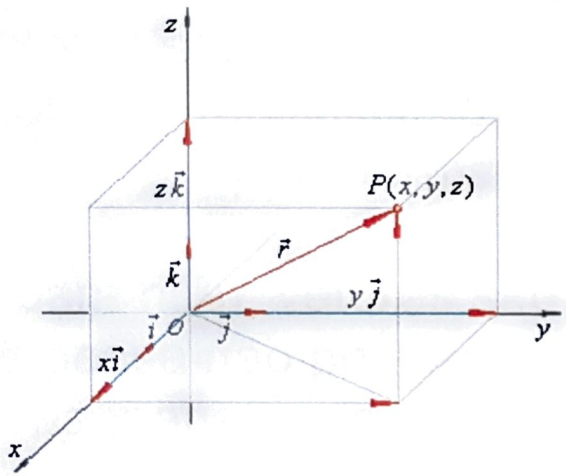
1 x 2 = 2
2 x 1 = 2

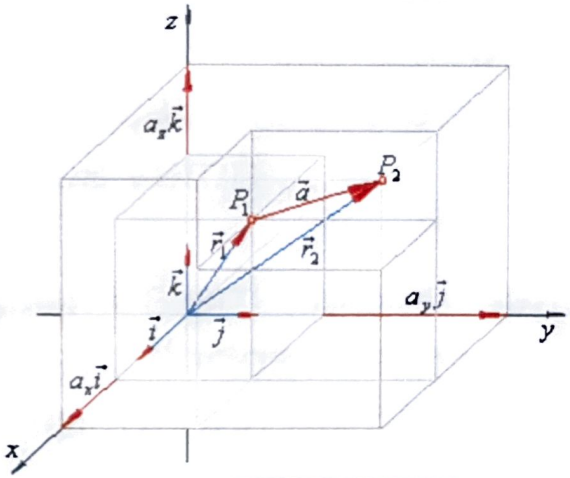
2+2=4

Dot Product

Fim

Vetores e Espaço Vetorial em Data Science





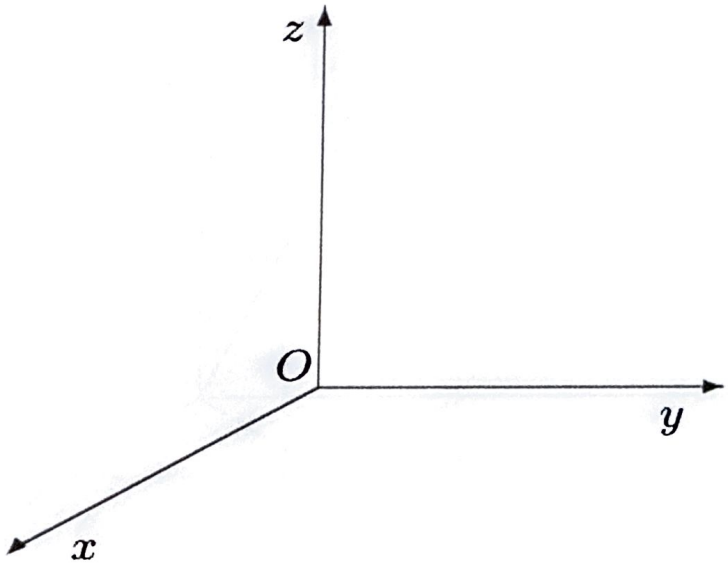
O Que São Vetores?

Abstratamente, vetores são objetos que podem ser somados (para formar vetores) e que podem ser multiplicados por escalares (ou seja, números), ta para formar novos vetores.

Concretamente (para nós), vetores são pontos em algum espaço de dime finita. Embora você possa não pensar em seus dados como vetores, eles s boa maneira de representar dados numéricos.

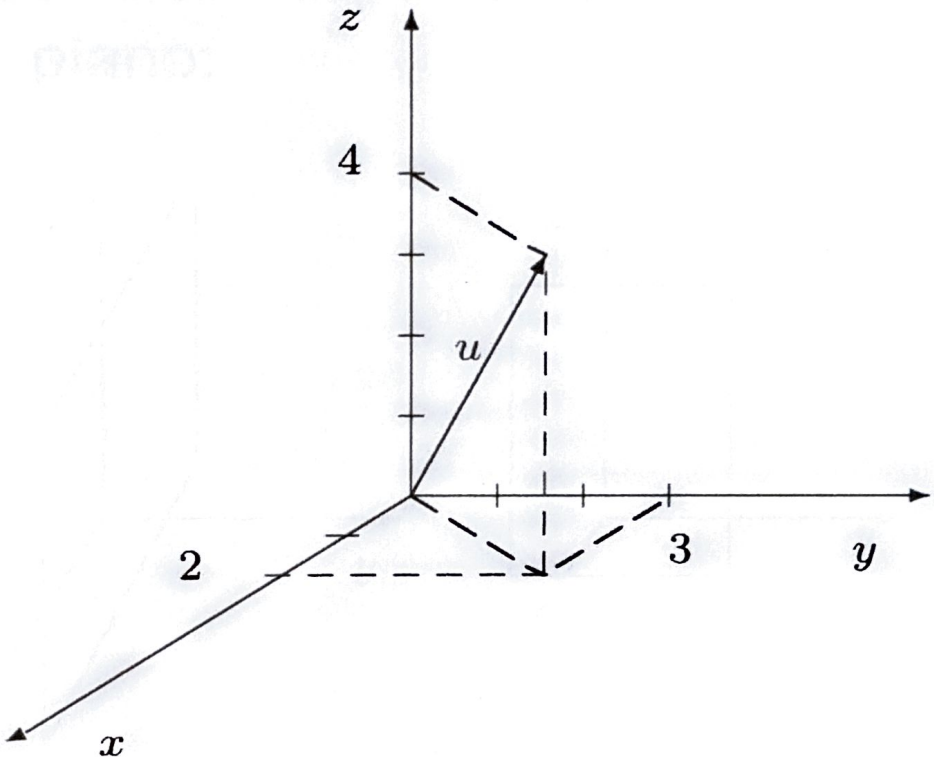
Vetores no Espaço

Eixo x é o eixo das abscissas.
Eixo y é o eixo das ordenadas.
Eixo z é o eixo das cotas.



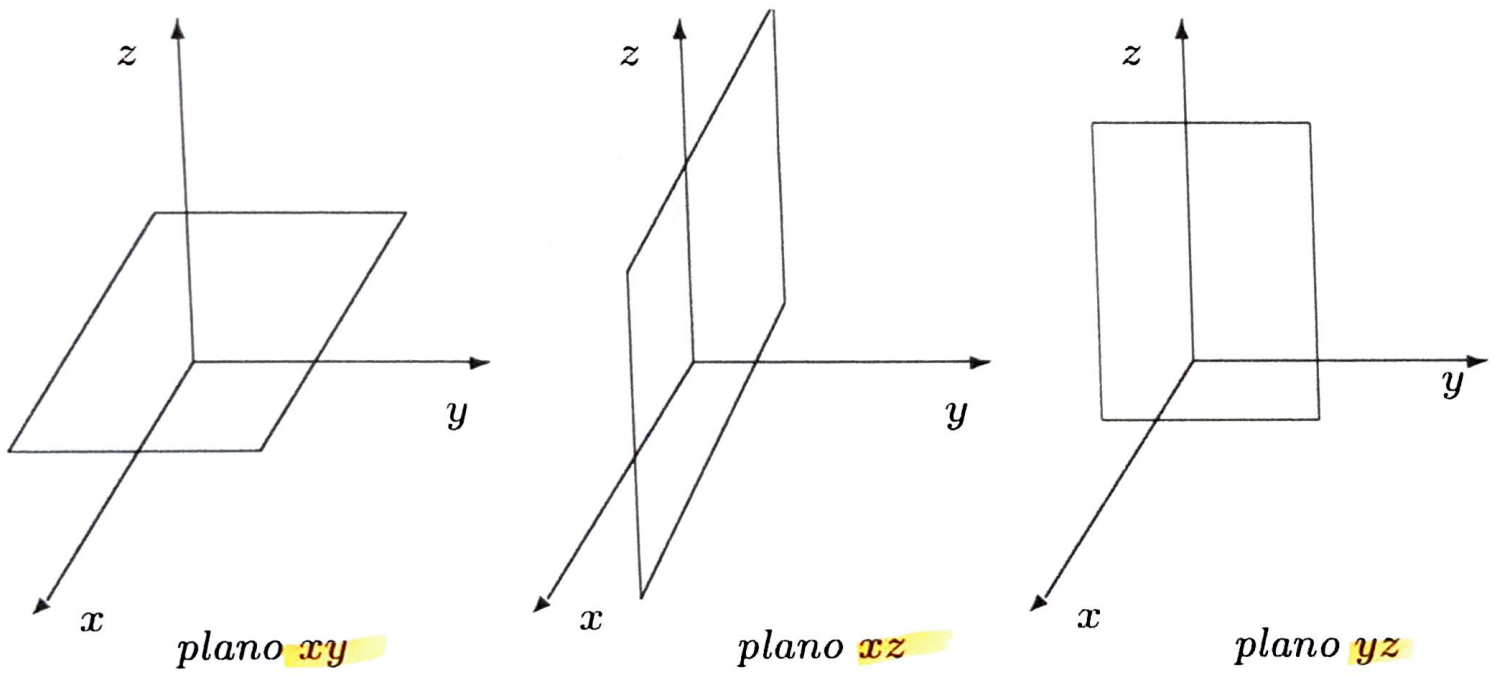
Vetores no Espaço

u = (2, 3, 4)



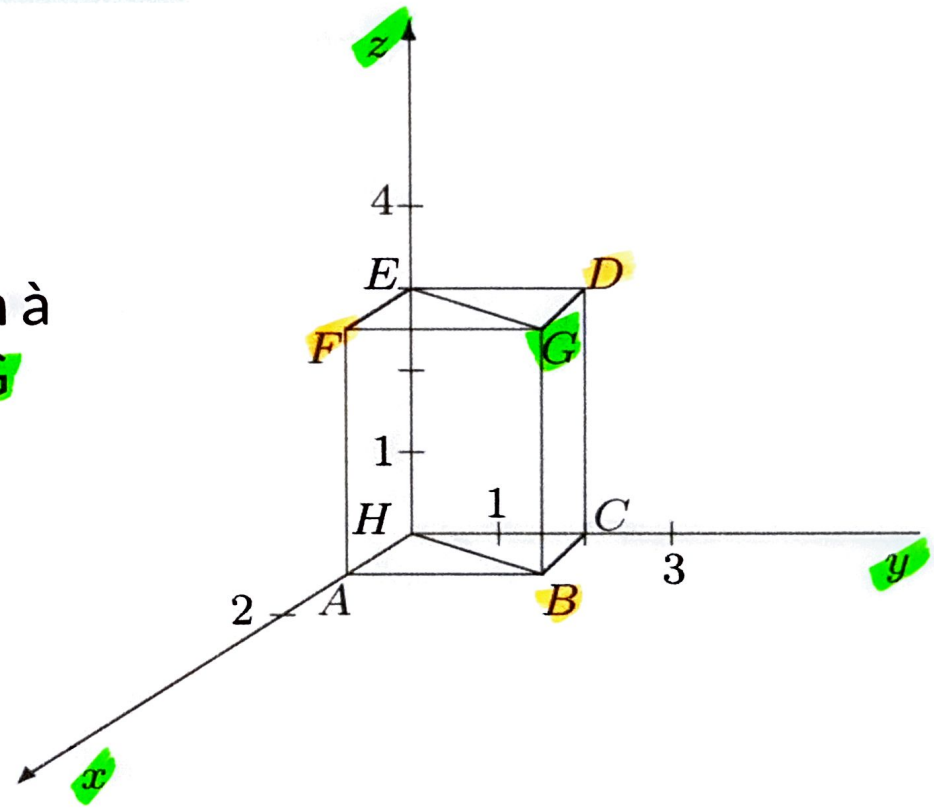
Vetores no Espaço

Cada dupla de eixos corresponde a um plano:



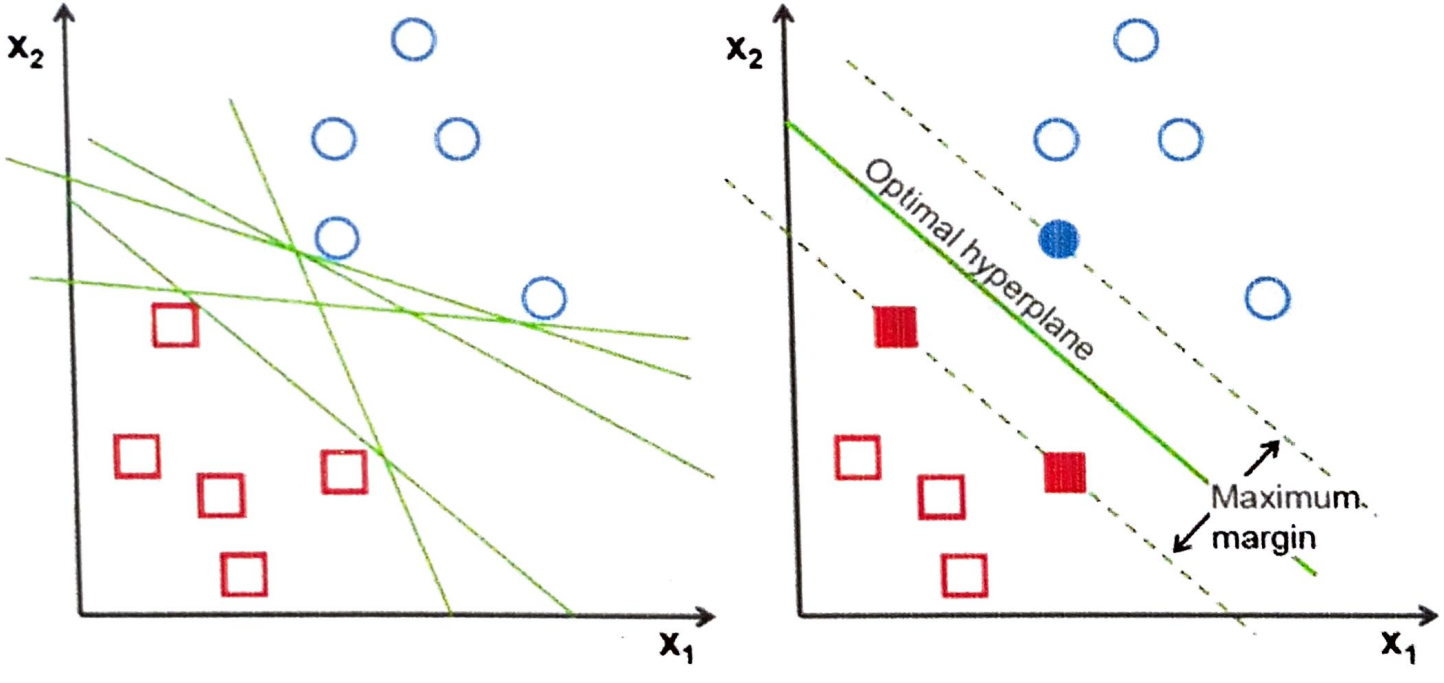
Vetores no Espaço

Os pontos B, D e F correspondem à **Projeção Ortogonal** do ponto G sobre os planos xy, yz e xz.



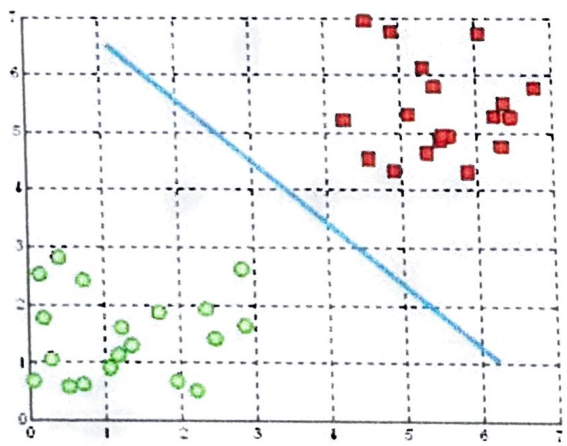
Estudo de Caso 1

Algoritmo Support Vector Machines

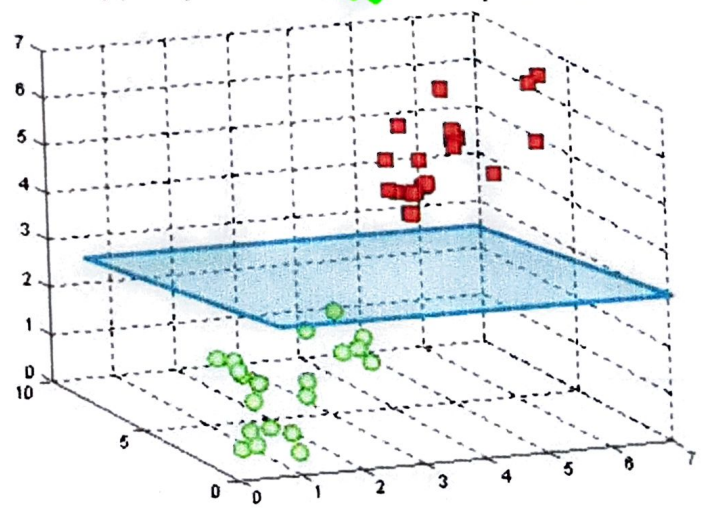


Estudo de Caso 1

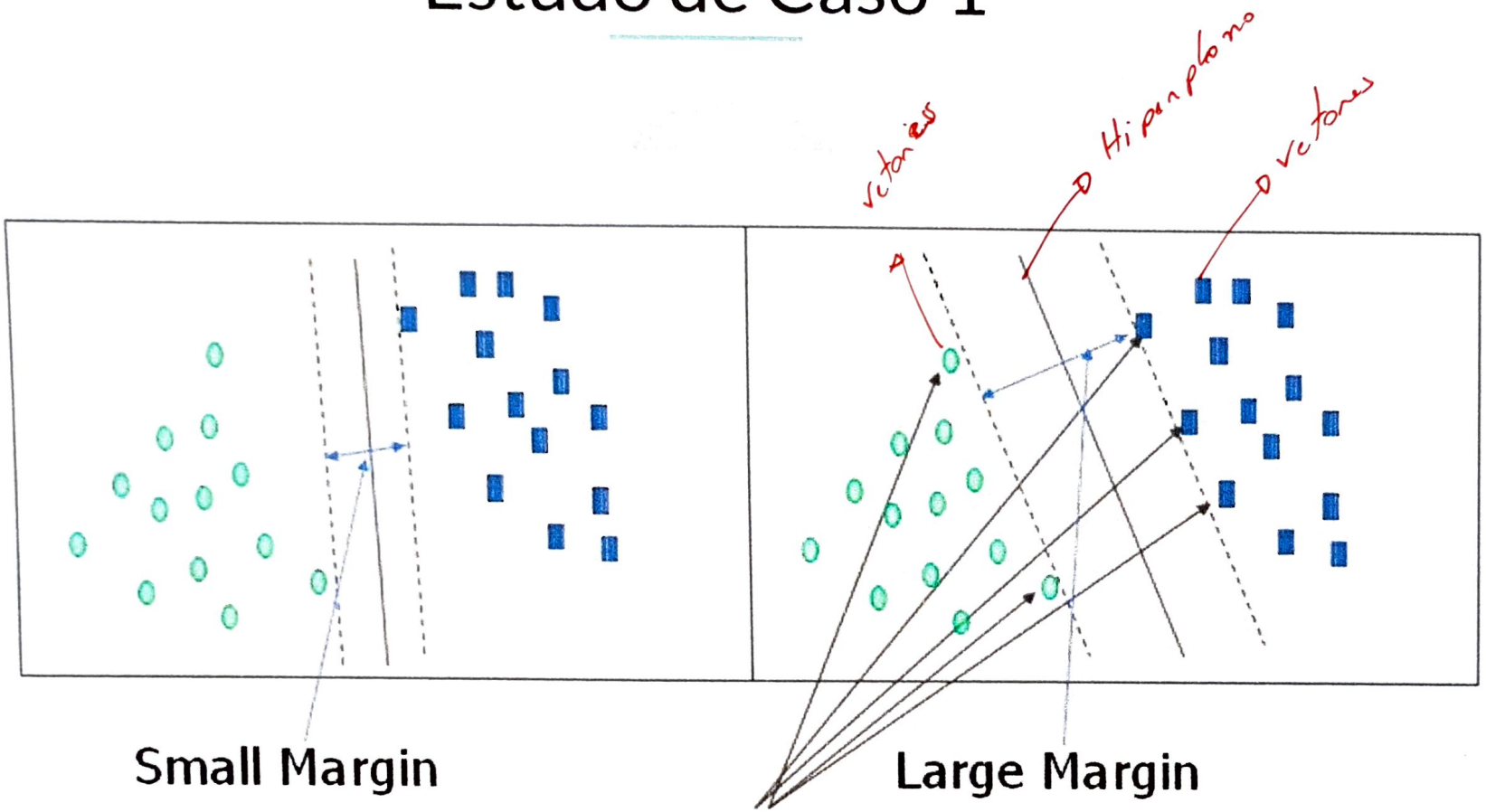
A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



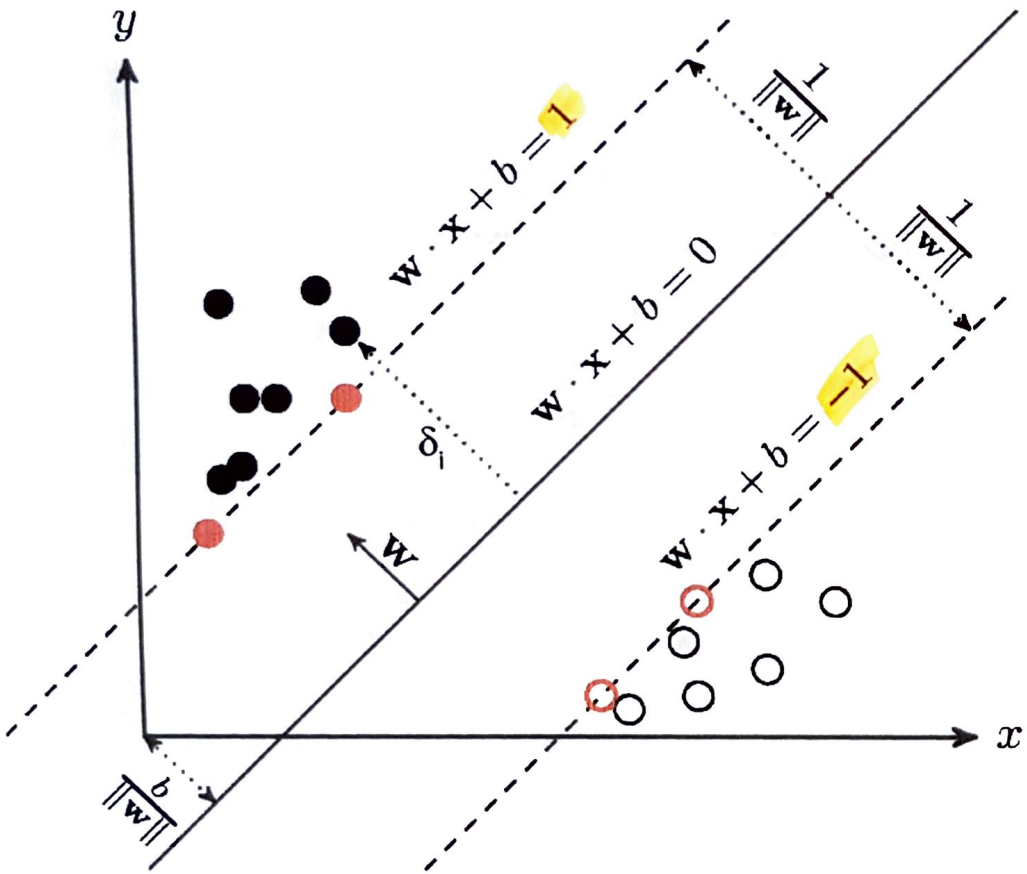
Estudo de Caso 1



Support Vectors

os vetores ajudam a encontrar as Margens para então encontrar o Hiperplano

Estudo de Caso 1



O objetivo do algoritmo é encontrar uma linha (hyperplano) que melhor separa os pontos de dados, atribuindo pesos a cada entrada de dados e somando um valor de bias (viés)

Estudo de Caso 1

Função de Custo (Função de Perda)

No algoritmo SVM, procuramos maximizar a margem entre os pontos de dados e o hiperplano. A função de perda que ajuda a maximizar a margem é a hinge loss

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Estudo de Caso 1

Função de Custo (Função de Perda)

O custo é 0 se o valor previsto e o valor real forem do mesmo sinal. Se não forem, calculamos o valor da perda. Também adicionamos um parâmetro de regularização à função de custo objetivo do parâmetro de regularização é equilibrar a maximização e a perda de margem Depois de adicionar o parâmetro de regularização, a função de custo é exibida abaixo

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Estudo de Caso 1

Função de Custo (Função de Perda)

Agora que temos a função de perda, tomamos derivadas parciais em relação aos pesos para encontrar os gradientes. Usando os gradientes, podemos atualizar nossos pesos

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\big(1 - y_i\langle x_i, w\rangle\big)_+ \quad = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Estudo de Caso 1

Quando não há erro de classificação, ou seja, nosso modelo prevê corretamente a classe do nosso ponto de dado, precisamos apenas atualizar o gradiente do parâmetro de regularização.

$$w = w - \alpha \cdot (2\lambda w)$$

Quando há um erro de classificação, ou seja, nosso modelo comete um erro na previsão da classe do nosso ponto de dado, incluímos a perda juntamente com o parâmetro de regularização para executar a atualização do gradiente.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

modulo 4

Em Python uma lista não é um vetor

Precisamos usar Array ou Numpy

Data Science Academy

Matemática Para Machine Learning

Solução - Lista Extra de Exercícios - Capítulo 3

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

A maioria das pessoas considera a Matemática e a Física, assustadoras bestas das quais é melhor manter a distância. Computadores, no entanto, podem nos ajudar a domar a complexidade e a aritmética tediosa das manipulações associadas a esses assuntos. De fato, Matemática e Física são muito mais acessíveis, uma vez que você tem o poder de computadores do seu lado.

Divirta-se!!

In [1]:

```
from scipy import *
from scipy.linalg import *
from matplotlib.pyplot import *
%matplotlib inline
```

Considere os vetores v1 e v2 abaixo para resolver os exercícios de 1 a 5.

In [2]:

```
# Vetores
v1 = array([1., 2., 3.])
v2 = array([2, 0, 1.])
```

Exercício 1 - Multiplique o vetor V1 por 3 e divida o vetor V2 por 4.

In [3]:

```
# Solução V1
print(3 * v1)
```

```
[3. 6. 9.]
```

In [4]:

```
# Solução V2
print(v2 / 4)
```

V1 = 1, 2, 3
V2 = 2, 0, 1

```
[0.5  0.   0.25]
```

2 0 1

Exercício 2 - Faça uma combinação linear entre eles. Multiplique o vetor 1 por 3 e multiplique o vetor v2 por 5. Então some os resultados.

In [5]:

```
# Solução
print(3*v1 + 5*v2)
```

```
[13.  6. 14.]
```

Exercício 3 - Encontre o produto escalar entre os vetores v1 e v2.

Dica: O produto escalar é um único número resultado da multiplicação entre todos os elementos dos vetores. Considerando os vetores v1 e v2 acima, o resultado do produto escalar é 5. Crie seu código que encontra esse resultado.

In [6]:

```
# Solução
dot(v1, v2)
```

Out[6]:

```
5.0
```

Produto escalar entre os vetores V1 e V2
Calculo de todos os elementos dos vetores

Exercício 4 - Faça uma multiplicação element-wise entre os vetores v1 e v2.

↳ multiplico cada elemento de um vetor pelo correspondente do outro vetor

In [7]:

```
# Solução
v1 * v2
```

1	2	3
2	0	1
2	0	3

Out[7]:

```
array([2., 0., 3.])
```

Exercício 5 - Encontre o cosseno do vetor v1.

In [8]:

```
# Solução
cos(v1)
```

Out[8]:

```
array([ 0.54030231, -0.41614684, -0.9899925 ])
```

Exercício 6 - Considere o vetor v e a matriz M abaixo. Multiplique o vetor pela matriz.

A ordem dos objetos na multiplicação é relevante?

In [9]:

```
v = array([1.,2.])
M = array([[1.,2],[3.,4]])
```

Multiplicação entre vetores e Matrizes é importante

In [10]:

```
# Solução
dot(v, M)
```

Out[10]:

```
array([ 7., 10.])
```

Resultados diferente

In [12]:

```
dot(M, v)
```

Out[12]:

```
array([ 5., 11.])
```

A ordem altera o resultado

Exercício 7 - Usando a função solve() do pacote linalg do scipy, encontre a solução do sistema de equações lineares representado pelos 2 arrays abaixo.

In [13]:

```
import scipy.linalg as sl
A = array([[1., 2.],
           [3., 4.]])

b = array([1., 4.])
```

In [14]:

```
# Solução
x = sl.solve(A,b)
print(x)
```

```
[ 2.  -0.5]
```

Exercício 8 - Explique porque a operação abaixo com os mesmos objetos A e b do exercício anterior apresenta resultado diferente.

In [15]:

```
# Solução
dot(A, b)
```

Out[15]:

```
array([ 9., 19.])
```

Resposta: O resultado é diferente porque no exercício 7 estamos resolvendo um sistema de equações e os arrays são apenas representações destas equações. Aqui no exercício 8 estamos tratando os arrays apenas como listas de números e realizando multiplicação entre eles, o que é bem diferente de resolver equações.

Exercício 9 - Desafio

Considerando a matriz esparsa abaixo, pesquise pela função do pacote scipy que permite comprimir esta matriz.

Nota: Matriz esparsa é uma matriz preenchida com mais valores 0 do que outros elementos. Comprimir a matriz significa exatamente remover os valores 0. Matrizes esparsas e compressão são amplamente utilizados em Machine Learning.

Matrizes esparsas têm aplicações em problemas de engenharia e física (por exemplo, o método das malhas para resolução de circuitos elétricos ou sistemas de equações lineares). Também têm aplicação em computação: armazenamento de dados (e.g., planilhas eletrônicas).

In [16]:

```
A = array([[1,0,2,0],[0,0,0,0],[3.,0.,0.,0.],[1.,0.,0.,4.]])
print(A)
```

```
[[1. 0. 2. 0.]
 [0. 0. 0. 0.]
 [3. 0. 0. 0.]
 [1. 0. 0. 4.]]
```

In [17]:

```
# Solução
import scipy.sparse as sp
A = array([[1,0,2,0],[0,0,0,0],[3.,0.,0.,0.],[1.,0.,0.,4.]])
AS = sp.csr_matrix(A)
AS.data
```

Out[17]:

```
array([1., 2., 3., 1., 4.])
```

Exercício 10 - Super Desafio

Abaixo você encontra uma classe Python que descreve uma rede neural artificial apenas com operações matemáticas. Não se deixe impressionar pelo código, é bem mais simples do que parece!!

Seu trabalho é completar a função sigmoide com o código necessário para executar a operação matemática para esta que é uma função de ativação e define a saída de cada neurônio matemático.

In [ ]:

Perceptron

```python
# Imports
import random
import numpy as np

# Classe Network
class Network(object):

    def __init__(self, sizes):
        """A lista `sizes` contém o número de neurônios nas
         respectivas camadas da rede. Por exemplo, se a lista
         for [2, 3, 1] então será uma rede de três camadas, com o
         primeira camada contendo 2 neurônios, a segunda camada 3 neurônios,
         e a terceira camada 1 neurônio. Os bias e pesos para a
         rede são inicializados aleatoriamente, usando uma distribuição Gaussiana com média
         Note que a primeira camada é assumida como uma camada de entrada, e por convenção
         não definimos nenhum bias para esses neurônios, pois os bias são usados
         na computação das saídas das camadas posteriores."""

        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes[1:])]

    def feedforward(self, a):
        """Retorna a saída da rede se `a` for input."""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a

    def SGD(self, training_data, epochs, mini_batch_size, eta, test_data=None):
        """Treinar a rede neural usando mini-batch stochastic
        gradient descent. O `training_data` é uma lista de tuplas
         `(x, y)` representando as entradas de treinamento e as
         saídas. Os outros parâmetros não opcionais são
         auto-explicativos. Se `test_data` for fornecido, então a
         rede será avaliada em relação aos dados do teste após cada
         época e progresso parcial impresso. Isso é útil para
         acompanhar o progresso, mas retarda as coisas substancialmente."""

        training_data = list(training_data)
        n = len(training_data)

        if test_data:
            test_data = list(test_data)
            n_test = len(test_data)

        for j in range(epochs):
            random.shuffle(training_data)
            mini_batches = [training_data[k:k+mini_batch_size] for k in range(0, n, mini_ba

            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)

            if test_data:
                print("Epoch {} : {} / {}".format(j,self.evaluate(test_data),n_test));
            else:
                print("Epoch {} finalizada".format(j))
```

```python
def update_mini_batch(self, mini_batch, eta):
    """Atualiza os pesos e bias da rede aplicando
     a descida do gradiente usando backpropagation para um único mini lote.
     O `mini_batch` é uma lista de tuplas `(x, y)`, e `eta` é a taxa de aprendizado."""

    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]

    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]

    self.weights = [w-(eta/len(mini_batch))*nw for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb for b, nb in zip(self.biases, nabla_b)]

def backprop(self, x, y):
    """Retorna uma tupla `(nabla_b, nabla_w)` representando o
     gradiente para a função de custo C_x. `nabla_b` e
     `nabla_w` são listas de camadas de matrizes numpy, semelhantes
     a `self.biases` e `self.weights`."""

    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]

    # Feedforward
    activation = x

    # Lista para armazenar todas as ativações, camada por camada
    activations = [x]

    # Lista para armazenar todos os vetores z, camada por camada
    zs = []

    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)

    # Backward pass
    delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())

    # Aqui, l = 1 significa a última camada de neurônios, l = 2 é a segunda e assim por
    for l in range(2, self.num_layers):
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)

def evaluate(self, test_data):
    """Retorna o número de entradas de teste para as quais a rede neural
     produz o resultado correto. Note que a saída da rede neural
     é considerada o índice de qualquer que seja
     neurônio na camada final que tenha a maior ativação."""
```

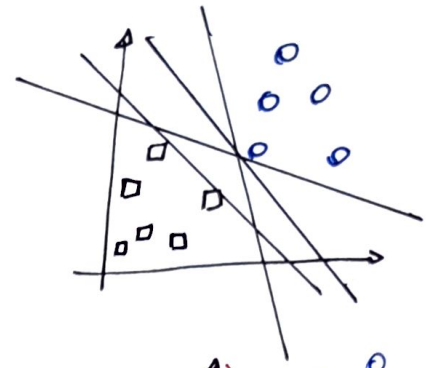
```
        test_results = [(np.argmax(self.feedforward(x)), y) for (x, y) in test_data]
        return sum(int(x == y) for (x, y) in test_results)

    def cost_derivative(self, output_activations, y):
        """Retorna o vetor das derivadas parciais."""
        return (output_activations-y)

# Função de Ativação Sigmóide
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

Fim

Support Vectors Machine

```python
# Estudo de Caso 1 - SVMs

####### Importando e Preparando o Dataset #######

# Import pandas
import pandas as pd

# Carrega o dataset
df = pd.read_csv('iris.csv')

# Remove a coluna ID
df = df.drop(['Id'],axis=1)

# Define a variável alvo
target = df['Species']

# Cria um conjunto
s = set()

# Valores possíveis da variável target
for val in target:
    s.add(val)

# Converte para lista
s = list(s)

# Separa em dados de treino e de teste
rows = list(range(100,150))
df = df.drop(df.index[rows])


####### Plot dos Dados #######

# Import matplotlib
import matplotlib.pyplot as plt

# Define x e y (entrada e saída)
x = df['SepalLengthCm']
y = df['PetalLengthCm']

# Dados referentes a uma classe de flores
setosa_x = x[:50]
setosa_y = y[:50]

# Dados referentes a outra classe de flores
versicolor_x = x[50:]
versicolor_y = y[50:]

# Plot
plt.figure(figsize=(8,6))
plt.scatter(setosa_x,setosa_y,marker='+',color='green')
plt.scatter(versicolor_x,versicolor_y,marker='_',color='red')
plt.show()


####### Algoritmo SVM #######

from sklearn.utils import shuffle
from sklearn.cross_validation import train_test_split
import numpy as np

# Descarta o restante dos recursos e extraia os valores de destino
df = df.drop(['SepalWidthCm','PetalWidthCm'],axis=1)
Y = []
target = df['Species']

for val in target:
    if(val == 'Iris-setosa'):
```

o ID pode afetar o modelo



optimal Hiperplane

maximum margem

deixa apenas dois tipos de flores
as ultimos 50 variaveis são excluidos

os pontos mais próximos as margens são chamados de Support Vectors

do 0 ao 50

do 50 até o fim do vetor lista

excluindo 2 colunas que não vai usar

define a variavel que será o target

```
            Y.append(-1)
        else:
            Y.append(1)
    df = df.drop(['Species'],axis=1)
    X = df.values.tolist()

    # Embaralhe e divida os dados de treinamento e testes
    X, Y = shuffle(X,Y)
    x_train = []
    y_train = []
    x_test = []
    y_test = []

    x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size=0.9)

    # Converte tudo para array NumPy
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    x_test = np.array(x_test)
    y_test = np.array(y_test)

    # Ajusta o shape
    y_train = y_train.reshape(90,1)
    y_test = y_test.reshape(10,1)

    # Dados de treino das flores 1 e 2
    train_f1 = x_train[:,0]
    train_f2 = x_train[:,1]

    # Ajusta o shape dos dados das flores 1 e 2
    train_f1 = train_f1.reshape(90,1)
    train_f2 = train_f2.reshape(90,1)

    # Cria o vetor de pesos
    w1 = np.zeros((90,1))
    w2 = np.zeros((90,1))

    # Hiperparâmetros
    epochs = 1
    alpha = 0.0001

    # Treinamento
    while(epochs < 10000):

        y = w1 * train_f1 + w2 * train_f2

        prod = y * y_train

        print(epochs)

        count = 0

        for val in prod:
            if(val >= 1):
                cost = 0
                w1 = w1 - alpha * (2 * 1/epochs * w1)
                w2 = w2 - alpha * (2 * 1/epochs * w2)

            else:
                cost = 1 - val
                w1 = w1 + alpha * (train_f1[count] * y_train[count] - 2 * 1/epochs * w1)
                w2 = w2 + alpha * (train_f2[count] * y_train[count] - 2 * 1/epochs * w2)
            count += 1
        epochs += 1
```

eixo 0 e 1
X Y
linha
coluna

converte todos os dados p/ array Numpy

90% dos dados p/ treino

10% para teste

Ajusta o shape para poder fazer operações com os objetos

Cria arrays com zeros para receber os pesos

nº de treinamentos

Peso

2º peso

taxa de aprendizagem
Para que você possa mudar durante o valor dos pesos

Calculo o custo

SVM

```python
####### Avalia o Modelo #######

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")


# Agora, cortamos os pesos, pois os dados de teste contêm apenas 10 pontos de dados
index = list(range(10,90))
w1 = np.delete(w1,index)
w2 = np.delete(w2,index)

# Ajusta o shape
w1 = w1.reshape(10,1)
w2 = w2.reshape(10,1)

# Extrair os recursos de dados de teste
test_f1 = x_test[:,0]
test_f2 = x_test[:,1]

test_f1 = test_f1.reshape(10,1)
test_f2 = test_f2.reshape(10,1)

# Previsões
y_pred = w1 * test_f1 + w2 * test_f2
predictions = []
for val in y_pred:
    if(val > 1):
        predictions.append(1)
    else:
        predictions.append(-1)

print("\nAcurácia nos Dados de Teste (Algoritmo Personalizado)",
accuracy_score(y_test, predictions))


# Testando agora com o algoritmo pronto do Scikit-Learn
clf = SVC(kernel='linear')
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print("\nAcurácia nos Dados de Teste (Algoritmo Pronto do
Scikit-Learn)",accuracy_score(y_test, y_pred))
print("\n")
```

→ Precisa ajustar os dados do mesmo forma que fez no treinamento

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 5

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

A maioria das pessoas considera a Matemática e a Física, assustadoras bestas das quais é melhor manter a distância. Computadores, no entanto, podem nos ajudar a domar a complexidade e a aritmética tediosa das manipulações associadas a esses assuntos. De fato, Matemática e Física são muito mais acessíveis, uma vez que você tem o poder de computadores do seu lado.

Divirta-se!!

Exercício 1 - Crie um algoritmo em Python para a multiplicação das duas matrizes abaixo. Seu algoritmo deve checar se as matrizes atendem às condições de multiplicação de matrizes estudadas neste capítulo.

ATENÇÃO: NÃO use o NumPy (seria fácil demais). O objetivo é você testar sua lógica de programação para operação com matrizes.

$$x \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \quad y \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 3 & 4 \end{bmatrix}$$

4x3 3x2

In [1]:

```python
# Solução

# Matrizes
x = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
y = [[1,2],[1,2],[3,4]]

def matrixmult (A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    if cols_A != rows_B:
        print("Não é possível multiplicar essas duas matrizes. Dimensões incorretas.")
        return

    # Crie a matriz de resultados como uma matriz nula
    # Dimensões devem ser rows_A x cols_B
    C = [[0 for row in range(cols_B)] for col in range(rows_A)]
    print("\nMatriz nula criada para receber o resultado da operação: ", C)

    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]
    return C

matriz_resultante = matrixmult(x, y)
print("\nMatriz resultante da multiplicação entre as duas matrizes: ", matriz_resultante)
```

Criando duas matrizes

Recebe as duas Matrizes

tamanho linhas

significa Colunas de uma Matriz

Se o tamanho de colunas A for diferente de linhas B

2 listcomprehension

loops alinhados → Percorre linhas de A e colunas de B adicionando

Percorre Matriz

Exercício 2 - Agora sim, usando NumPy, verifique se o algoritmo criado no exercício anterior realmente está correto validando o resultado com o que o NumPy oferece para multiplicação de matrizes.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$(1 \times 1) + (2 \times 1) + (3 + 3)$

$4 \times 1 + 5 \times 1 + 6 \times 3$

$7 \times 1 + 8 \times 1 + 9 \times 3$

$10 \times 1 + 11 \times 1 + 12 \times 3$

$1 \times 2 + 2 \times 2 + 3 \times 4$

$4 \times 2 + 5 \times 2 + 6 \times 4$

$7 \times 2 + 8 \times 2 + 9 \times 4$

$10 \times 2 + 11 \times 2 + 12 \times 4$

$$\begin{bmatrix} 12 & 18 \\ 27 & 42 \\ 42 & 66 \\ 57 & 90 \end{bmatrix}$$

todos as linhas pelo 1º coluna

Todos as linha pelo 2º Coluna

In [2]:

```
# Solução

import numpy as np

# Matrizes
x = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
y = [[1,2],[1,2],[3,4]]

# Converte as listas Python para matrizes NumPy
mx = np.matrix(x)
my = np.matrix(y)

# Resultado
mx * my
```

→ Numpy faz o calculo automaticamente

Out[2]:

```
matrix([[12, 18],
        [27, 42],
        [42, 66],
        [57, 90]])
```

Exercício 3 - Considerando as duas matrizes abaixo, crie um algoritmo para multiplicar as matrizes.

Dica: Use loops aninhados para compreender a regra matemática e traduzir isso em linguagem de programação.

In [3]:

```python
# Solução

# Matriz 3x3
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]

# Matriz 3x4
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]

# Matriz nula 3x4
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]

# Iteração pelas linhas de x
for i in range(len(X)):

    # Iteração pelas colunas de y
    for j in range(len(Y[0])):

        # Iteração pelas linhas de y
        for k in range(len(Y)):
            result[i][j] += X[i][k] * Y[k][j]

# Imprime o resultado
for r in result:
    print(r)
```

```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
```

colunas de X tem que ser igual linhas Y

	1	2	3
1	12	7	3
2	4	5	6
3	7	8	9

	1	2	3	4
1	5	8	1	2
2	6	7	3	0
3	4	5	9	1

$(12 \times 5) + (7 \times 6) + (3 \times 4)$
$(12 \times 8) + (7 \times 7) + (3 \times 5)$
$(12 \times 1) + (7 \times 3) + (3 \times 9)$
$(12 \times 2) + (7 \times 0) + (3 \times 1)$

	1	2	3	
	$(4 \times 5) + (5 \times 6) + (6 \times 4)$			1
	$(4 \times 8) + (5 \times 7) + (6 \times 5)$			2
	$(4 \times 1) + (5 \times 3) + (6 \times 9)$			3
	$(4 \times 2) + (5 \times 0) + (6 \times 1)$			4

$(7 \times 5) + (8 \times 6) + (9 \times 4)$
$(7 \times 8) + (8 \times 7) + (9 \times 5)$
$(7 \times 1) + (8 \times 3) + (9 \times 1)$
$(7 \times 2) + (8 \times 0) + (9 \times 1)$

Exercício 4 - Realize a mesma operação anterior, mas desta vez usando list comprehension.

In [4]:

```
# Solução

# Matriz 3x3
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]

# Matriz 3x4
Y = [[5,8,1,2],
     [6,7,3,0],
     [4,5,9,1]]

# Resultado 3x4
result = [[sum(a*b for a,b in zip(X_row,Y_col)) for Y_col in zip(*Y)] for X_row in X]

# Imprime o resultado
for r in result:
    print(r)
```

```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
```

3º a executar

o zip são tuplas de valores

2º a executar

Asterisco indica todos

1º a executar

a b a b a b

5x12 + 6x7 + 4x3

8x12 + 7x7 + 5x3

b

$$\begin{bmatrix} 5 & 6 & 4 \\ 8 & 7 & 5 \\ 1 & 3 & 9 \\ 2 & 0 & 1 \end{bmatrix}$$

Y a

b

$$\begin{bmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

X b

Exercício 5 - Considere as duas matrizes abaixo, x e y, e as duas diferentes soluções apresentadas para a multiplicação entre elas. Para cada execução, nós usamos o operador mágico %timeit para medir o tempo de execução. Você saberia explicar porque o tempo de execução é muito diferente?

Dica: Essa costuma ser uma pergunta em entrevista para posição de Cientista de Dados.

```
For X_row in X:
    For Y_col in zip (* Y):
        For a,b in zip(X_row, Y_col):
            result = + (a * b)
```

In [5]:

```
# Solução

import numpy as np

x = [[1,2,3],[4,5,6],[7,8,9]]
y = [[1,2,3],[1,2,3],[3,4,5]]

def matrixmult (A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    if cols_A != rows_B:
        print("Não é possível multiplicar essas duas matrizes. Dimensões incorretas.")
        return

    # Crie a matriz de resultados como uma matriz nula
    # Dimensões devem ser rows_A x cols_B
    C = [[0 for row in range(cols_B)] for col in range(rows_A)]

    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]
    return C

# Tempo de execução na multiplicação customizada
print("\nTempo de execução na multiplicação customizada:")
%timeit matrixmult(x, y)
```

↳ só funciona no jupter Notebook

```
# Tempo de execução na multiplicação via NumPy
print("\nTempo de execução na multiplicação via NumPy:")
%timeit np.multiply(x, y)
```

```
Tempo de execução na multiplicação customizada:
9.32 µs ± 135 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

Tempo de execução na multiplicação via NumPy:
5.38 µs ± 25.4 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Resposta: As funções do NumPy são otimizadas para performance e não precisamos "reinventar a roda", tentando criar o que já existe!

Exercício 6 - Calcule o determinante da matriz abaixo, usando NumPy.

In [6]:

```
# Solução

import numpy

# Matriz
M = [[1, 2], [3, 4]]

# Determinante
print (numpy.linalg.det(M))
```

```
-2.0000000000000004
```

Exercício 7 - Calcule a inversa da matriz abaixo usando NumPy.

In [7]:

```
# Solução

x = [[2,3],[4,5]]

m = np.matrix(x)

# Solução 1
print("\nSolução 1:")
m.I
```

invanse

```
Solução 1:
```

Out[7]:

```
matrix([[-2.5,  1.5],
        [ 2. , -1. ]])
```

In [8]:

```
# Solução 2
print("\nSolução 2:")
np.linalg.inv(m)
```

```
Solução 2:
```

Out[8]:

```
matrix([[-2.5,  1.5],
        [ 2. , -1. ]])
```

Exercício 8 - O TensorFlow é o principal framework para Deep Learning. Usando uma de suas funções, encontre a inversa da matriz abaixo.

Referência: https://www.tensorflow.org/api_guides/python/math_ops#Matrix_Math_Functions (https://www.tensorflow.org/api_guides/python/math_ops#Matrix_Math_Functions)

Dica: Pode ser necessário instalar o TensorFlow com o comando abaixo (descomente a linha e execute a célula)

In [9]:

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /anaconda3/lib/python3.6/site-p
ackages (1.10.1)
Requirement already satisfied: protobuf>=3.6.0 in /anaconda3/lib/python3.6/s
ite-packages (from tensorflow) (3.6.1)
Requirement already satisfied: astor>=0.6.0 in /anaconda3/lib/python3.6/site
-packages (from tensorflow) (0.7.1)
Requirement already satisfied: gast>=0.2.0 in /anaconda3/lib/python3.6/site-
packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio>=1.8.6 in /anaconda3/lib/python3.6/sit
e-packages (from tensorflow) (1.15.0)
Requirement already satisfied: six>=1.10.0 in /anaconda3/lib/python3.6/site-
packages (from tensorflow) (1.11.0)
Requirement already satisfied: numpy<=1.14.5,>=1.13.3 in /anaconda3/lib/pyth
on3.6/site-packages (from tensorflow) (1.14.3)
Requirement already satisfied: setuptools<=39.1.0 in /anaconda3/lib/python3.
6/site-packages (from tensorflow) (39.1.0)
Requirement already satisfied: tensorboard<1.11.0,>=1.10.0 in /anaconda3/li
b/python3.6/site-packages (from tensorflow) (1.10.0)
Requirement already satisfied: wheel>=0.26 in /anaconda3/lib/python3.6/site-
packages (from tensorflow) (0.31.1)
Requirement already satisfied: absl-py>=0.1.6 in /anaconda3/lib/python3.6/si
te-packages (from tensorflow) (0.4.1)
Requirement already satisfied: termcolor>=1.1.0 in /anaconda3/lib/python3.6/
site-packages (from tensorflow) (1.1.0)
Requirement already satisfied: markdown>=2.6.8 in /anaconda3/lib/python3.6/s
ite-packages (from tensorboard<1.11.0,>=1.10.0->tensorflow) (2.6.11)
Requirement already satisfied: werkzeug>=0.11.10 in /anaconda3/lib/python3.
6/site-packages (from tensorboard<1.11.0,>=1.10.0->tensorflow) (0.14.1)
distributed 1.21.8 requires msgpack, which is not installed.
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [10]:

tensorflow não é mais veloz do que o Numpy

```
# Solução

import tensorflow as tf
import warnings
warnings.filterwarnings('ignore')

x = [[2,3],[4,5]]

x_cast = tf.cast(x, tf.float64)
x_final = tf.linalg.inv(x_cast)
print(x_final)
```

conversão de tipo de valor do contrário o tensorflow não aceitaria

```
/anaconda3/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compil
etime version 3.5 of module 'tensorflow.python.framework.fast_tensor_util' d
oes not match runtime version 3.6
  return f(*args, **kwds)
/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: C
onversion of the second argument of issubdtype from `float` to `np.floating`
is deprecated. In future, it will be treated as `np.float64 == np.dtype(floa
t).type`.
  from ._conv import register_converters as _register_converters

Tensor("MatrixInverse:0", shape=(2, 2), dtype=float64)
```

Exercício 9 (Desafio) - Crie um algoritmo de Machine Learning de Regressão Linear Simples, usando NumPy, que mostre a relação entre duas variáveis x e y (que são criadas abaixo usando valores aleatórios). Seu script deve conter as operações matemáticas de multiplicação de matrizes estudadas neste capítulo.

Dica: Deixamos algumas linhas de código para ajudá-lo nesta tarefa.

In [11]:

```
# Solução

# Imports
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Variáveis x e y
num_points = 200
X_entrada = np.linspace(-2, 2, num_points)
y_saida = X_entrada + np.random.normal(0, 1, num_points)

# Print shapes
print(np.shape(X_entrada))
print(np.shape(y_saida))

# Plot
plt.plot(X_entrada, y_saida, 'r*', label='Dados')
plt.legend(loc="upper left", bbox_to_anchor=(1,1))
```

ignora warnings

exemplo de regressão quanto maior o N= de quartos maior o preço da casa

limite de pontos = 200

Gera valores aleatórios

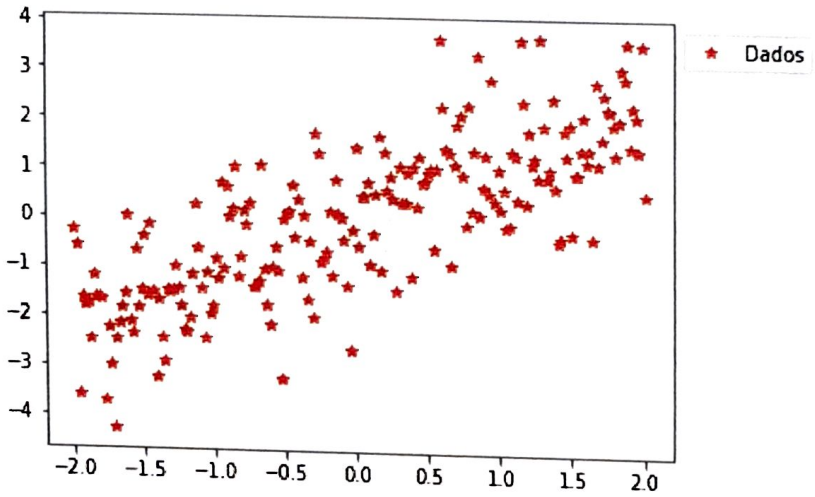
Gero as dimensões dos objetos

```
(200,)
(200,)
```

dois vetores representando X e y

Out[11]:

```
<matplotlib.legend.Legend at 0xb2b5175f8>
```



In [12]:

```
X_entrada
```

Out[12]:

```
array([-2.        , -1.9798995 , -1.95979899, -1.93969849, -1.91959799,
       -1.89949749, -1.87939698, -1.85929648, -1.83919598, -1.81909548,
       -1.79899497, -1.77889447, -1.75879397, -1.73869347, -1.71859296,
       -1.69849246, -1.67839196, -1.65829146, -1.63819095, -1.61809045,
       -1.59798995, -1.57788945, -1.55778894, -1.53768844, -1.51758794,
       -1.49748744, -1.47738693, -1.45728643, -1.43718593, -1.41708543,
       -1.39698492, -1.37688442, -1.35678392, -1.33668342, -1.31658291,
       -1.29648241, -1.27638191, -1.25628141, -1.2361809 , -1.2160804 ,
       -1.1959799 , -1.1758794 , -1.15577889, -1.13567839, -1.11557789,
       -1.09547739, -1.07537688, -1.05527638, -1.03517588, -1.01507538,
       -0.99497487, -0.97487437, -0.95477387, -0.93467337, -0.91457286,
       -0.89447236, -0.87437186, -0.85427136, -0.83417085, -0.81407035,
       -0.79396985, -0.77386935, -0.75376884, -0.73366834, -0.71356784,
       -0.69346734, -0.67336683, -0.65326633, -0.63316583, -0.61306533,
       -0.59296482, -0.57286432, -0.55276382, -0.53266332, -0.51256281,
       -0.49246231, -0.47236181, -0.45226131, -0.4321608 , -0.4120603 ,
       -0.3919598 , -0.3718593 , -0.35175879, -0.33165829, -0.31155779,
       -0.29145729, -0.27135678, -0.25125628, -0.23115578, -0.21105528,
```

Poderia ser nº de quartos

In [13]:

```
y_saida
```

Out[13]:

```
array([-2.60469233e-01, -6.12297797e-01, -3.61560726e+00, -1.64811686e+00,
       -1.81052008e+00, -1.76981567e+00, -2.48438321e+00, -1.19245864e+00,
       -1.62415259e+00, -1.63206332e+00, -1.66968132e+00, -3.73680887e+00,
       -2.23721829e+00, -2.99604188e+00, -4.28852755e+00, -2.49103995e+00,
       -2.17479730e+00, -1.84656259e+00, -1.54698561e+00, -2.79644710e-03,
       -2.11923426e+00, -2.37378742e+00, -6.89579407e-01, -1.83179516e+00,
       -1.46678268e+00, -4.05586300e-01, -1.58728072e+00, -1.38862351e-01,
       -1.50618696e+00, -3.25594816e+00, -1.69774613e+00, -2.44802278e+00,
       -2.91728178e+00, -1.46264372e+00, -1.53675938e+00, -1.46895612e+00,
       -9.83063432e-01, -1.46019283e+00, -1.78560361e+00, -2.28464552e+00,
       -2.30512483e+00, -2.03008859e+00, -1.13992887e+00,  2.57918124e-01,
       -6.43484331e-01, -1.43510430e+00, -2.43811666e+00, -1.13787851e+00,
       -1.97080721e+00, -1.79656286e+00, -8.26145323e-01, -1.23487049e+00,
        7.05223696e-01, -1.05539584e+00,  5.92288255e-01,  2.09326761e-02,
        1.56774091e-01,  1.01019586e+00, -1.21702696e+00, -7.88567622e-01,
        1.30302347e-01, -1.47676569e-01,  3.01690254e-01, -1.39037409e+00,
       -1.40759719e+00, -1.26889108e+00,  1.04422646e+00, -1.05073311e+00,
       -1.77638582e+00, -2.17552032e+00, -1.00234630e+00, -6.08887628e-01,
```

Poderia ser o preço das casas

In [14]:

```
# Algoritmo

# Matriz de 1s para servir como as previsões
ones = np.ones(num_points)

# Convertendo as variáveis para matrizes NumPy
X_matrix = np.matrix(np.column_stack((X_entrada, ones)))  # Matriz aumentada (explicada nas
y_matrix = np.transpose(np.matrix(y_saida))

# Print shapes
print(np.shape(X_matrix))
print(np.shape(y_matrix))

# Encontrando a transposta das matrizes e aplicando o dot product
XtX = X_matrix.transpose().dot(X_matrix)
Xty = X_matrix.transpose().dot(y_matrix)

# Print shapes
print(np.shape(XtX))
print(np.shape(Xty))

# Inversa
solution = np.linalg.inv(XtX).dot(Xty).tolist()
print("\nCoeficientes da Regressão:")
print(solution)

# Coeficientes
w, b = solution

# Previsões
y_pred = w[0]*X_entrada + b[0]
print("\n", y_pred)

# Plot
plt.plot(X_entrada, y_saida, 'r*', label='Dados')
plt.plot(X_entrada, y_pred, 'b', label='Linha de Regressão - Previsões NumPy')
plt.legend(loc="upper left", bbox_to_anchor=(1,1))
```

200

Matriz aumentado
Une o x_entrada com ones

para poder multiplicar as matrizes

Converte y_saida
Para uma Matriz transposta

ajuda a controlar o Byes

transpose
inverte colunas por linhas

matriz inversa

multiplicação de Matrizes

Fórmula do regressão

converte para tipo lista

módulo de algebra linear

Peso

Byes

$A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 4 & 6 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

Para Multiplicar
Poderia ser feito a transpose tanto A quant B

```
(200, 2)
(200, 1)
(2, 2)
(2, 1)

Coeficientes da Regressão:
[[1.0787248699273968], [0.011504377438197004]]

 [-2.14594536e+00 -2.12426245e+00 -2.10257954e+00 -2.08089663e+00
 -2.05921371e+00 -2.03753080e+00 -2.01584789e+00 -1.99416498e+00
 -1.97248207e+00 -1.95079915e+00 -1.92911624e+00 -1.90743333e+00
 -1.88575042e+00 -1.86406751e+00 -1.84238459e+00 -1.82070168e+00
 -1.79901877e+00 -1.77733586e+00 -1.75565295e+00 -1.73397004e+00
 -1.71228712e+00 -1.69060421e+00 -1.66892130e+00 -1.64723839e+00
 -1.62555548e+00 -1.60387256e+00 -1.58218965e+00 -1.56050674e+00
 -1.53882383e+00 -1.51714092e+00 -1.49545800e+00 -1.47377509e+00
 -1.45209218e+00 -1.43040927e+00 -1.40872636e+00 -1.38704344e+00
 -1.36536053e+00 -1.34367762e+00 -1.32199471e+00 -1.30031180e+00
 -1.27862888e+00 -1.25694597e+00 -1.23526306e+00 -1.21358015e+00
```

In [15]:

```
# Solução

# Import
import tensorflow as tf

# Variáveis x e y
num_points = 200
X_entrada = np.linspace(-2, 2, num_points)
y_saida = X_entrada + np.random.normal(0, 1, num_points)
```

Geração de dados de forma aleatória

```
# Matrizes (tensores com o TF)
X_tensor = tf.constant(X_matrix)
y_tensor = tf.constant(y_matrix)
```

Define as duas Matrizes (tensores)

```
# Transposta
XtX_tensor = tf.matmul(tf.transpose(X_tensor), X_tensor)
Xty_tensor = tf.matmul(tf.transpose(X_tensor), y_tensor)
```

Faz a transposta das duas matrizes

```
# Solução
solution_tf = tf.matmul(tf.matrix_inverse(XtX_tensor), Xty_tensor)
```

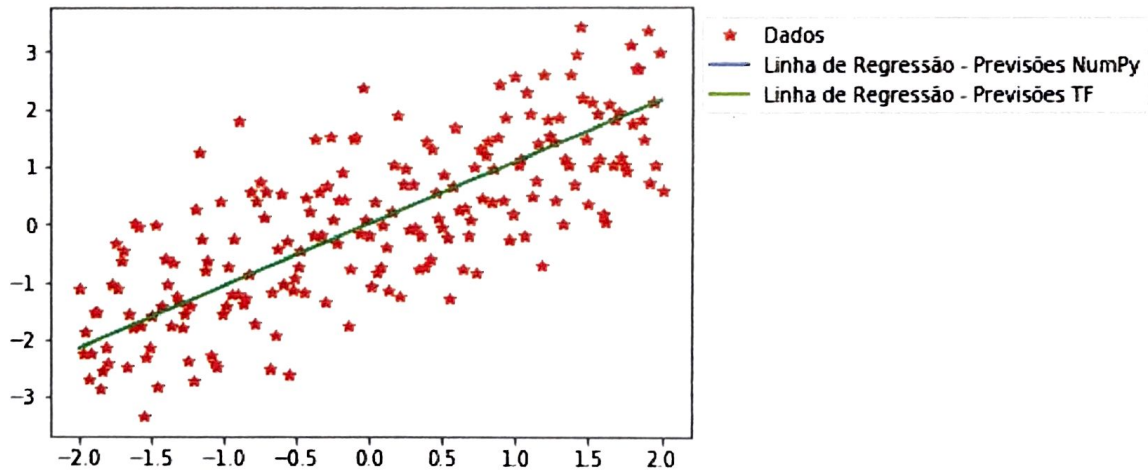
Solução

pego a matriz inversa de X e multiplica por Y

```
# Abre a sessão TF
sess = tf.Session()
soln_tf = sess.run(solution_tf).tolist()

# Encontra os coeficeintes
w_tf, b_tf = soln_tf

# Previsões
y_pred_tf = w_tf[0]*X_entrada + b_tf[0]

# Plot
plt.plot(X_entrada, y_saida, 'r*', label='Dados')
plt.plot(X_entrada, y_pred, 'b', label='Linha de Regressão - Previsões NumPy')
plt.plot(X_entrada, y_pred_tf, 'g', label='Linha de Regressão - Previsões TF')
plt.legend(loc="upper left", bbox_to_anchor=(1,1))
```

Out[15]:

```
<matplotlib.legend.Legend at 0xb2cb26400>
```



Data Science Academy

Matemática Para Machine Learning

Estudo de Caso 2 - Operações com Matrizes em Redes Neurais Artificiais

O diagrama abaixo mostra uma rede simples, um único neurônio chamado Perceptron.

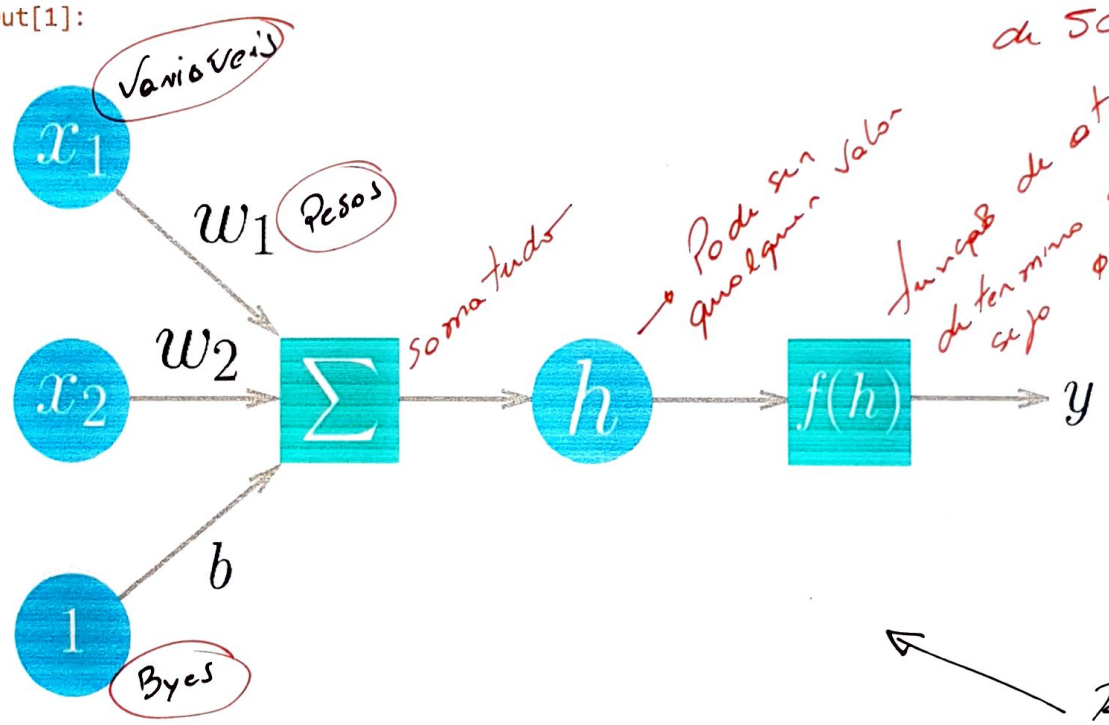
A combinação linear dos pesos, entradas e bias forma a entrada h, que passa pela função de ativação f(h), resultando a saída final do perceptron, identificada por y.

In [1]:

```
from IPython.display import Image
Image(url = 'simple-neuron.png')
```

Criado na década de 50

Out[1]:



Representa 1 neurônio.

Os círculos representam dados e os quadrados representam operações.

Deep learning são 3 bilhões de Neurônios

In [2]:

```
from IPython.display import Image
Image(url = 'step-function.gif')
```

Out[2]:

$$f(h) = \begin{cases} 0 & \text{if } h < 0 \\ 1 & \text{if } h \geq 0 \end{cases}$$

← Se o valor de H for menor que ∅ o valor é zero

← Se o valor de H for menor ou igual a ∅ a saída vai ser 1

A unidade de saída retorna o resultado de f(h), onde h é a entrada para a unidade de saída:

A parte interessante sobre esta arquitetura e o que possibilita a construção de modelos de redes neurais é que a função de ativação, f(h) pode ser qualquer função.

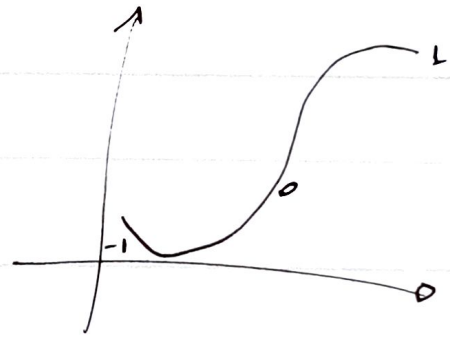
Estudaremos diversas funções de ativação, como as funções logísticas (muitas vezes chamadas de sigmóides), tanh, ReLu e softmax. Abaixo, usaremos a função sigmóide.

In [3]:

```
import numpy as np
```

In [4]:

```
# Função de ativação
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```



In [5]:

```
from IPython.display import Image
Image(url = 'sigmoid-formula.png')
```

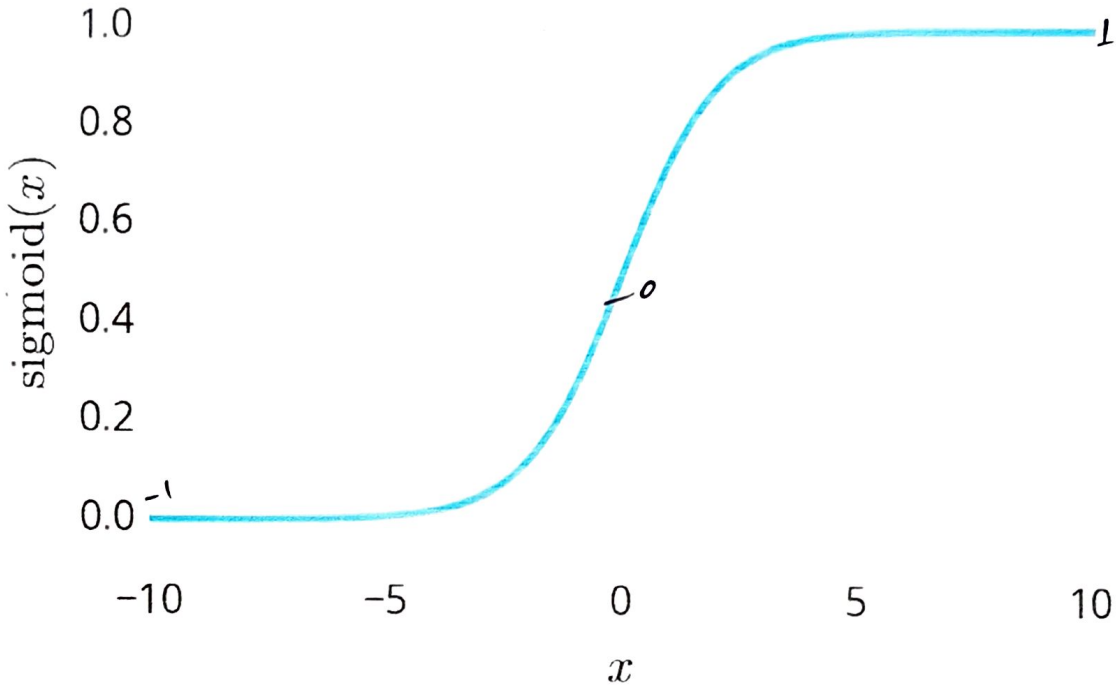
Out[5]:

$$\text{sigmoid}(x) = 1/(1 + e^{-x})$$

In [6]:

```
from IPython.display import Image
Image(url = 'sigmoid.png')
```

Out[6]:



A função sigmóide gera valores entre 0 e 1, e como uma saída pode ser interpretada como uma probabilidade de sucesso. A função sigmóide, como função de ativação, resulta na mesma formulação que a regressão logística.

Nesse momento, o perceptron deixa de receber esse nome e começa a ser chamado de rede neural. No caso de redes simples como esta, as redes neurais não oferecem qualquer vantagem em relação aos modelos lineares gerais, como a regressão logística.

In [7]:

```
# Gerando dados para o input, pesos e bias
inputs = np.array([0.7, -0.3])
pesos = np.array([0.1, 0.8])
bias = -0.1
```

x_1 x_2

→ esse é o valor depois de normalizado

→ Pesos, você ainda não sabe qual é o peso ideal

https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html
(https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html)

In [8]:

```
# Gerando a saída - Sem a função de ativação, não seria possível ter valores com saída entr
output = np.dot(pesos, inputs) + bias
```

igual valor de H

→ o Resultado será um único valor escalar que vem somado com Bias

In [9]:

```
print('Saída Sem Função de Ativação:')
print(output)
```

```
Saída Sem Função de Ativação:
-0.27
```

In [10]:

Aplicando a função de ativação.

```
# Gerando a saída (treinamento)
output = sigmoid(np.dot(pesos, inputs) + bias)
```

In [11]:

```
print('Saída Com Função de Ativação:')
print(output)
```

```
Saída Com Função de Ativação:
0.4329070950345457
```

Representa 1

Fim

Ainda não foi calculado a taxa de erro
Nem a derivada da função.

Data Science Academy - Matemática Para Machine Learning

Transposta

Obter a transposição de uma matriz é realmente fácil em NumPy. Basta acessar seu atributo T. Há também a função transpose() que retorna a mesma coisa, mas raramente você verá isso usado em qualquer lugar porque digitar T é muito mais fácil.

In [1]:

```
import numpy as np
```

In [2]:

```
m = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
m
```

Out[2]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [3]:

```
m.T
```

Out[3]:

```
array([[ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11],
       [ 4,  8, 12]])
```

NumPy faz isso sem realmente mover qualquer dado na memória - ele simplesmente muda a maneira como ele indexa a matriz original - por isso é bastante eficiente.

No entanto, isso também significa que você precisa ter cuidado com a forma como você modifica objetos, porque eles compartilham os mesmos dados. Por exemplo, com a mesma matriz m acima, vamos fazer uma nova variável m_t que armazene a transposição de m. Então, veja o que acontece se modificarmos um valor em m_t:

In [4]:

```
m_t = m.T
m_t[3][1] = 200
```

In [5]:

```
m_t
```

Out[5]:

```
array([[  1,   5,   9],
       [  2,   6,  10],
       [  3,   7,  11],
       [  4, 200,  12]])
```

In [6]:

```
m
```

Out[6]:

```
array([[  1,   2,   3,   4],
       [  5,   6,   7, 200],
       [  9,  10,  11,  12]])
```

Observe como ele modificou a transposição e a matriz original também! Isso porque eles estão compartilhando a mesma cópia de dados. Então lembre-se de considerar a transposição apenas como uma visão diferente de sua matriz, ao invés de uma matriz diferente, inteiramente.

Exemplo em Redes Neurais

Digamos que você tenha as seguintes duas matrizes, chamadas inputs e pesos:

In [7]:

```
inputs = np.array([[-0.27,  0.45,  0.64, 0.31]])
inputs
```

Out[7]:

```
array([[-0.27,  0.45,  0.64,  0.31]])
```

In [8]:

```
inputs.shape
```

Out[8]:

```
(1, 4)
```

In [9]:

```
pesos = np.array([[0.02, 0.001, -0.03, 0.036], [0.04, -0.003, 0.025, 0.009], [0.012, -0.045
```

```
pesos
```

Out[9]:

```
array([[ 0.02 ,  0.001, -0.03 ,  0.036],
       [ 0.04 , -0.003,  0.025,  0.009],
       [ 0.012, -0.045,  0.28 , -0.067]])
```

In [10]:

```
pesos.shape
```

Out[10]:

```
(3, 4)
```

Os pesos são multiplicados aos inputs nas camadas das redes neurais, logo precisamos multiplicar pesos por inputs. Uma multiplicação de matrizes. Isso é muito fácil com o NumPy.

In [11]:

```
np.matmul(inputs, pesos)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-d190cbc25737> in <module>()
----> 1 np.matmul(inputs, pesos)

ValueError: shapes (1,4) and (3,4) not aligned: 4 (dim 1) != 3 (dim 0)
```

Ops, o que deu errado? Se você fez a aula de multiplicação de matrizes, então você já viu esse erro antes. As matrizes estão shapes incompatíveis porque o número de colunas na matriz esquerda, 4, não é igual ao número de linhas na matriz direita, 3.

Como resolvemos isso? Podemos usar a transposta da matriz:

In [12]:

```
np.matmul(inputs, pesos.T)
```

Out[12]:

```
array([[-0.01299,  0.00664,  0.13494]])
```

Também funciona se você gerar a transposta da matriz de inputs e trocar a ordem dos parâmetros na função:

In [13]:

```
np.matmul(pesos, inputs.T)
```

Out[13]:

```
array([[-0.01299],
       [ 0.00664],
       [ 0.13494]])
```

As duas respostas são transpostas uma da outra, de modo que a multiplicação que você usa realmente depende apenas da forma (shape) que deseja para a saída.

Fim

Data Science Academy - Matemática Para Machine Learning

Multiplicação de Matrizes com NumPy

Multiplicação Element-wise

Já vimos a multiplicação de matrizes com element-wise. Apenas para relembrar:

In [1]:

```
import numpy as np
```

In [2]:

```
# Definimos uma matriz
m = np.array([[1,2,3],[4,5,6]])
m
```

Out[2]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [3]:

```
# Usamos o operador de multiplicação para multiplicar cada elemento da matriz por 0.50.
# Isso vai gerar uma nova matriz n.
n = m * 0.50
n
```

Out[3]:

```
array([[0.5, 1. , 1.5],
       [2. , 2.5, 3. ]])
```

In [4]:

```
# Multiplicamos as duas matrizes, element-wise
m * n
```

Out[4]:

```
array([[ 0.5,  2. ,  4.5],
       [ 8. , 12.5, 18. ]])
```

Matrix Product

Para encontrar o produto da matriz, você usa a função matmul() do NumPy. O shape das matrizes precisa ser compatível.

In [5]:

```
a = np.array([[1,2,3,4],[5,6,7,8]])
a
```

Out[5]:

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [6]:

```
a.shape
```

Out[6]:

```
(2, 4)
```

In [7]:

```
b = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
b
```

Out[7]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [8]:

```
b.shape
```

Out[8]:

```
(4, 3)
```

observe que o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz.
Relebrando as regras:

• O número de colunas na matriz esquerda deve ser igual ao número de linhas na matriz direita.

• A matriz de respostas sempre possui o mesmo número de linhas que a matriz esquerda e o mesmo número de colunas que a matriz direita.

• Ordem importa aqui. Multiplicar A • B não é o mesmo que multiplicar B • A.

• Os dados na matriz esquerda devem ser organizados como linhas, enquanto os dados na matriz direita devem ser organizados como colunas.

In [9]:

```
matriz_reposta = np.matmul(a, b)
matriz_reposta
```

Out[9]:

```
array([[ 70,  80,  90],
       [158, 184, 210]])
```

In [10]:

```
matriz_reposta.shape
```

Out[10]:

```
(2, 3)
```

In [11]:

```
# Se as matrizes tiverem shape incompatível, você recebe a mensagem de erro abaixo.
# Veja que A x B não é a mesma coisa que B x A na multiplicação de matrizes.
np.matmul(b, a)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-5cbc6b44f7d2> in <module>()
      1 # Se as matrizes tiverem shape incompatível, você recebe a mensagem
 de erro abaixo.
      2 # Veja que A x B não é a mesma coisa que B x A na multiplicação de m
atrizes.
----> 3 np.matmul(b, a)

ValueError: shapes (4,3) and (2,4) not aligned: 3 (dim 1) != 2 (dim 0)
```

Função dot() do NumPy

Às vezes, você pode ver a função dot() do NumPy em lugares onde você esperaria um matmul. Acontece que os resultados das funções dot() e matmul() são os mesmos se as matrizes forem de duas dimensões.

In [12]:

```
a = np.array([[1,2],[3,4]])
a
```

Out[12]:

```
array([[1, 2],
       [3, 4]])
```

In [13]:

```
np.dot(a,a)
```

Out[13]:

```
array([[ 7, 10],
       [15, 22]])
```

In [14]:

```
# Podemos chamar a função dot() diretamente do objeto ndarray
a.dot(a)
```

Out[14]:

```
array([[ 7, 10],
       [15, 22]])
```

In [15]:

```
np.matmul(a,a)
```

Out[15]:

```
array([[ 7, 10],
       [15, 22]])
```

Embora essas funções retornem os mesmos resultados para dados bidimensionais, você deve ter cuidado com o que você escolher ao trabalhar com outras formas de dados. Seguem abaixo referências nas documentações oficiais:

NumPy matmul() - https://docs.scipy.org/doc/numpy/reference/generated/numpy.matmul.html#numpy.matmul (https://docs.scipy.org/doc/numpy/reference/generated/numpy.matmul.html#numpy.matmul)

NumPy dot() - https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html)

Fim

Data Science Academy - Matemática Para Machine Learning

Operações Element-wise com NumPy

Primeiro, como você faria usando Python puro

Suponha que você tenha uma lista de números e que você deseja adicionar 5 a cada item da lista. Sem NumPy, você pode fazer algo como isto:

In [1]:

```
# Lista de valores
valores = [1, 2, 3, 4, 5]
```

In [2]:

```
# Looop for para adicionar 5 a cada elemento da lista
for i in range(len(valores)):
    valores[i] += 5
```

In [3]:

```
print(valores)
```

```
[6, 7, 8, 9, 10]
```

Isso faz sentido, mas é muito código para escrever e é mais lento, pois é Python puro.

Agora sim, usando NumPy

In [4]:

```
import numpy as np
```

In [5]:

```
valores = [1,2,3,4,5]
```

In [6]:

```
valores = np.array(valores) + 5
```

In [7]:

```
print(valores)
```

```
[ 6  7  8  9 10]
```

Bem mais simples, não? Perceba que usamos a função array() para converter o objeto Python para um objeto NumPy e então realizar o cálculo.

Mas isso pode ficar ainda mais simples. NumPy oferece funções prontas para operações matemáticas como essa. Por exemplo:

In [8]:

```
valores = [1,2,3,4,5]
```

In [9]:

```
valores = np.add(valores, 5)
```

In [10]:

```
print(valores)
```

```
[ 6  7  8  9 10]
```

NumPy facilita muito nosso trabalho e vários frameworks de Deep Learning se baseiam no NumPy.

Um outro exemplo de operação element-wise com escalares e objetos ndarrays. Digamos que você tenha um objeto chamado "valores" e você queira reutilizá-lo, mas primeiro você precisa definir todos os seus valores em zero. Fácil, basta multiplicar por zero e atribuir o resultado de volta ao objeto, assim:

In [11]:

```
valores *= 0
```

In [12]:

```
print(valores)
```

```
[0 0 0 0 0]
```

Operações Element-wise com Matrizes

As mesmas funções e operadores que trabalham com escalares e matrizes também funcionam com outras dimensões. Você só precisa se certificar de que os itens que você executa a operação possuem shapes compatíveis.

In [13]:

```
x = np.array([[1,3],[5,7]])
x
```

Out[13]:

```
array([[1, 3],
       [5, 7]])
```

In [14]:

```
y = np.array([[2,4],[6,8]])
y
```

Out[14]:

```
array([[2, 4],
       [6, 8]])
```

In [15]:

```
x + y
```

Out[15]:

```
array([[ 3,  7],
       [11, 15]])
```

Isso funciona porque é uma multiplicação de elementos entre duas matrizes de forma (shape) idêntica. Experimente alterar o shape de uma das matrizes e repetir a soma. Você receberá uma mensagem de erro.

Fim

Data Science Academy - Matemática Para Machine Learning

Operações Matemáticas com NumPy

Python é uma excelente linguagem de programação, mas ela pode ser lenta quando usada na sua forma básica. No entanto, ele permite que você acesse bibliotecas que executem código mais rápido escrito em linguagens como C. NumPy é uma dessas bibliotecas: fornece alternativas rápidas para operações matemáticas em Python e foi projetado para funcionar de forma eficiente com grupos de números - como matrizes.

NumPy é uma excelente biblioteca, sendo a base de quase todos os frameworks de aprendizagem de máquina em Python. Veremos agora algumas operações matemáticas essenciais para a construção de redes neurais e como realizar seu processamento com o NumPy.

In [1]:

```
import numpy as np
```

Tipos de Dados e Shapes

A maneira mais comum de trabalhar com números em NumPy é através de objetos ndarray. Eles são semelhantes às listas em Python, mas podem ter qualquer número de dimensões. Além disso, o ndarray suporta operações matemáticas rápidas, o que é exatamente o que queremos.

Como você pode armazenar qualquer número de dimensões, você pode usar ndarrays para representar qualquer um dos tipos de dados que abordamos antes: escalares, vetores, matrizes ou tensores.

Scalar

Escalares em NumPy são mais eficientes do que em Python. Em vez dos tipos básicos do Python como int, float, etc., o NumPy permite especificar tipos mais específicos, bem como diferentes tamanhos. Então, em vez de usar int em Python, você tem acesso a tipos como uint8, int8, uint16, int16 e assim por diante, ao usar o NumPy.

Esses tipos são importantes porque todos os objetos que você cria (vetores, matrizes, tensores) acabam por armazenar escalares. E quando você cria uma matriz NumPy, você pode especificar o tipo (mas cada item na matriz deve ter o mesmo tipo). Nesse sentido, os arrays NumPy são mais como arrays C do que as listas em Python.

Se você quiser criar uma matriz NumPy que contenha um escalar, usamos a função array do NumPy:

In [2]:

```
s = np.array(8)
```

Você ainda pode realizar matemática entre ndarrays, escalares NumPy e escalares Python normais, como veremos mais adiante.

Você pode ver o shape da matriz usando o atributo shape, conforme abaixo. Esse comando retorna um () vazio, indicando que este objeto é um escalar.

In [3]:

```
s.shape
```

Out[3]:

```
()
```

Mesmo que os escalares estejam dentro de arrays, você ainda os usa como um escalar normal, para operações matemáticas:

In [4]:

```
x = s - 2
```

In [5]:

```
x
```

Out[5]:

```
6
```

E x seria igual a 6. Se você verificar o tipo de x, vai perceber que é numpy.int64, pois você está trabalhando com tipos NumPy, e não com os tipos Python.

Mesmo os tipos escalares suportam a maioria das funções de matriz. Então você pode chamar x.shape e retornaria () porque tem zero dimensões, mesmo que não seja uma matriz. Se você tentar usar o objeto como um escalar Python normal, você obterá um erro.

In [6]:

```
type(x)
```

Out[6]:

```
numpy.int32
```

Vetores

Para criar um vetor, você passaria uma lista Python para a função array(), assim:

In [7]:

```
vec = np.array([1,2,3])
```

In [8]:

```
vec.shape
```

Out[8]:

```
(3,)
```

Se você verificar o atributo de shape do vetor, ele retornará um único número representando o comprimento unidimensional do vetor. No exemplo acima, vec.shape retorna (3,).

Agora que há um número, você pode ver que a forma é uma tupla com os tamanhos de cada uma das dimensões do ndarray. Para os escalares, era apenas uma tupla vazia, mas os vetores têm uma dimensão, então a tupla inclui um número e uma vírgula. (Python não entende (3) como uma tupla com um item, por isso requer a vírgula. Documentação oficial do Python 3.6 sobre Tuplas: https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences (https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences)

Você pode acessar um elemento dentro do vetor usando índices, como este abaixo (como você pode ver, em Python a indexaçã começa por 0 e o índice 1 representa o segundo elemento da matriz).

In [9]:

```
vec[1]
```

Out[9]:

```
2
```

NumPy também suporta técnicas avançadas de indexação. Por exemplo, para acessar os itens do segundo elemento em diante, você usaria:

In [10]:

```
vec[1:]
```

do posição 1 até o fim do vetor

Out[10]:

```
array([2, 3])
```

NumPy slicing é bastante poderoso, permitindo que você acesse qualquer combinação de itens em um ndarray. Documentação oficial sobre indexação e slicing de arrays: https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html (https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html)

Matrizes

Você cria matrizes usando a função de array() NumPy, exatamente como você fez com os vetores. No entanto, em vez de apenas passar uma lista, você precisa fornecer uma lista de listas, onde cada lista representa uma linha. Então, para criar uma matriz 3x3 contendo os números de um a nove, você poderia fazer isso:

In [11]:

```
m = np.array([[1,2,3], [4,5,6], [7,8,9]])
```

In [12]:

```
m.shape
```

Out[12]:

```
(3, 3)
```

Verificando o atributo shape, retornaria a tupla (3, 3) para indicar que a matriz tem duas dimensões, cada dimensão com comprimento 3.

Você pode acessar elementos de matrizes como vetores, mas usando valores de índice adicionais. Então, para encontrar o número 6 na matriz acima, você usaria:

In [13]:

```
m[1][2]
```

Out[13]:

```
6
```

Tensores

Os tensores são como vetores e matrizes, mas podem ter mais dimensões. Por exemplo, para criar um tensor 3x3x2x1, você pode fazer o seguinte:

In [14]:

```
t = np.array([[[[1],[2]],[[3],[4]],[[5],[6]]],[[[7],[8]],\
    [[9],[10]],[[11],[12]]],[[[13],[14]],[[15],[16]],[[17],[17]]]])
```

In [15]:

```
t.shape
```

Out[15]:

```
(3, 3, 2, 1)
```

Para acessar um elemento do tensor, usamos a indexação da mesma forma que fizemos com vetores e matrizes:

In [16]:

```
t[2][2][1][0]
```

Out[16]:

```
17
```

Alterando o Formato (shape)

Às vezes, você precisará alterar a forma de seus dados sem realmente alterar seu conteúdo. Por exemplo, você pode ter um vetor, que é unidimensional, mas precisa de uma matriz, que é bidimensional. Há duas maneiras pelas quais você pode fazer isso.

Digamos que você tenha o seguinte vetor:

In [17]:

```
vec = np.array([1,2,3,4])
```

In [18]:

```
vec.shape
```

Out[18]:

```
(4,)
```

Chamando vec.shape retornaria (4,). Mas e se você quiser uma matriz 1x4? Você pode conseguir isso com a função de reshape, assim:

In [19]:

```
x = vec.reshape(1,4)
```

In [20]:

```
x.shape
```

Out[20]:

```
(1, 4)
```

A função reshape pode ser usada para outras atividades com matrizes.

Fim

In [ ]:

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 6

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Exercício 1

- Use numpy.linspace () para gerar 1000 valores uniformemente espaçados entre 0 e 50 e atribua a x.
- Transforme x usando a equação y = 30x + 1000 e atribua o resultado a y1.
- Transforme x usando a equação y = 50x + 100 e atribua o resultado a y2.

Gere 2 gráficos de linha no mesmo plot:

- Um com x no eixo x e y1 no eixo y. Defina a cor da linha para "laranja".
- Um com x no eixo x e y2 no eixo y. Defina a cor da linha para "azul".

In [1]:

```
# Solução

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(0, 50, 1000)

y1 = 30*x + 1000
y2 = 50*x + 100

plt.plot(x, y1, c='orange')
plt.plot(x, y2, c='blue')
```

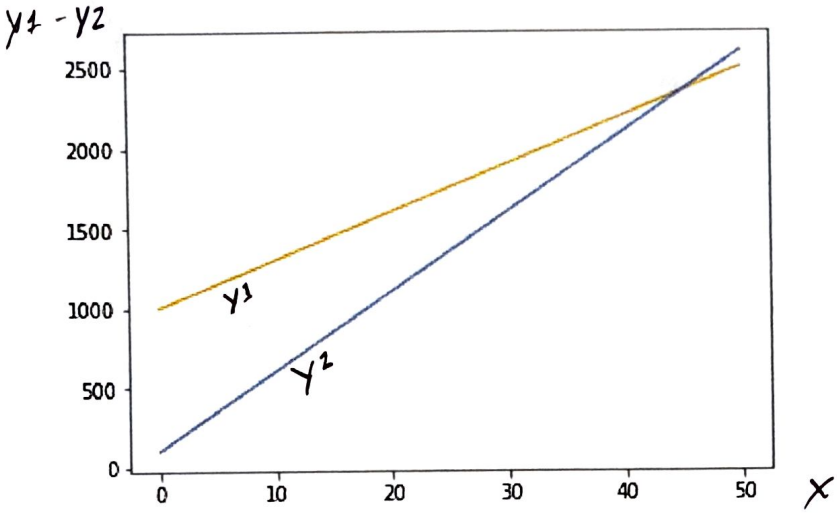
Gero 1000 Números entre ϕ e 50

1000 → 2.500

100 → 2.600

Out[1]:

```
[<matplotlib.lines.Line2D at 0x10a5099e8>]
```



Exercício 2

Considerando a matriz e vetor abaixo:

- matriz = np.array([[2,1],[1,1]])
- v1 = np.array([2,3])

Faça um dot product entre eles e grave no vetor v2. Crie um plot com os dois vetores, v1 e v2.

In [2]:

```
# Solução

import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Formatando o gráfico
plt.style.use('ggplot')
np.set_printoptions(suppress=True)
%precision 4
```

tipo do Gráfico

Nível de Precisão

```
# Função para criar o plot dos vetores
def plot_vectors(vs):
    """ Plotar vetores assumindo origem em (0,0) """
    n = len(vs)
    X, Y = np.zeros((n, 2))
    U, V = np.vstack(vs).T
    plt.quiver(X, Y, U, V, range(n), angles='xy', scale_units='xy', scale=1)
    xmin, xmax = np.min([U, X]), np.max([U, X])
    ymin, ymax = np.min([V, Y]), np.max([V, Y])
    xrng = xmax - xmin
    yrng = ymax - ymin
    xmin -= 0.05*xrng
    xmax += 0.05*xrng
    ymin -= 0.05*yrng
    ymax += 0.05*yrng
    plt.axis([xmin, xmax, ymin, ymax])
```

Poderia ser criado um arquivo .py para ser chamado import plots

Cria x e y como zeros

adicione o vs no vetor de zeros

quiver = função para desenhar vetores

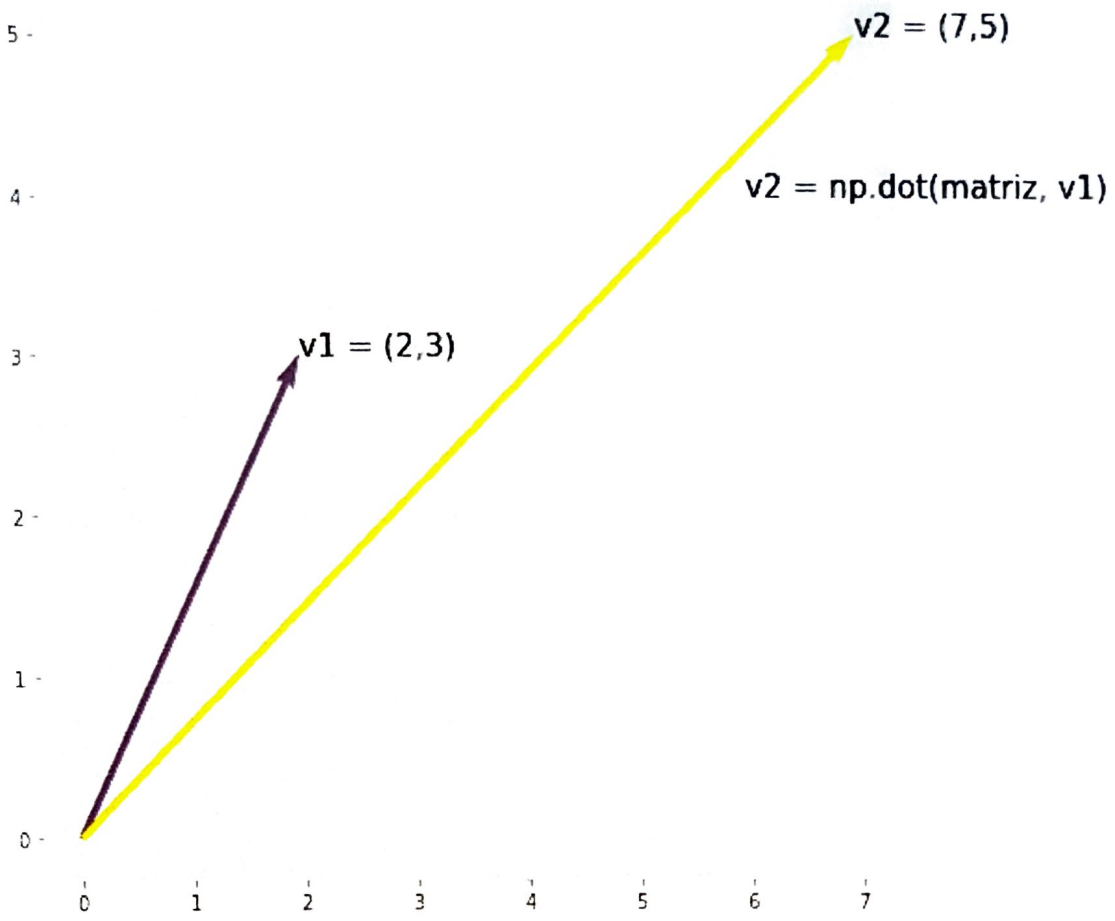
define os limites do Gráfico

```
# Matriz
matriz = np.array([[2,1],[1,1]])

# Vetor 1
v1 = np.array([2,3])

# Dot product entre a matriz e o vetor
v2 = np.dot(matriz, v1)

# Plot
plt.figure(figsize=(8,8))
plot_vectors([v1,v2])
plt.text(2,3,"v1 = (2,3)",fontsize=16)
plt.text(6,4,"v2 = np.dot(matriz, v1)", fontsize=16)
plt.text(v2[0],v2[1],"v2 = (7,5)",fontsize=16)
```

Legendas do Gráfico

Out[2]:

```
Text(7,5,'v2 = (7,5)')
```



Exercício 3

Sempre que um vetor contém 2 ou 3 elementos, podemos visualizá-lo em uma grade de coordenadas. Normalmente, um vetor é visualizado em uma grade de coordenadas usando setas, não usando coordenadas numéricas. Para um vetor contendo dois elementos, o primeiro valor descreve a coordenada x enquanto o segundo valor descreve a coordenada y. Os vetores são geralmente iniciados da origem até o ponto descrito pelo vetor.

As setas são usadas para visualizar vetores individuais porque enfatizam as duas propriedades principais de um vetor - direção e magnitude. A direção de um vetor descreve a maneira como ele aponta enquanto a magnitude descreve seu comprimento.

Podemos visualizar vetores no matplotlib usando a função pyplot.quiver().
https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.quiver
(https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.quiver)

Esta função tem 4 parâmetros requeridos: X, Y, U, V. Os parâmetros X e Y correspondem às coordenadas nais quais queremos que o vetor comece e U e V correspondam às coordenadas que queremos que o vetor termine.

Usando a função pyplot.quiver(), crie o plot para os vetores (2, 3), (-2, -3), (4, 3) e (1, 4). Cada vetor deve ter uma cor diferente.

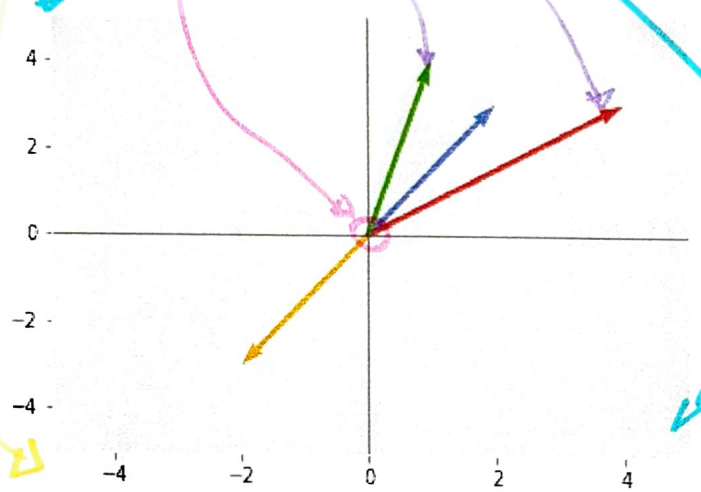
In [3]:

```
# Solução

# Imports
import matplotlib.pyplot as plt
import numpy as np

# Desenhamos o plano cartesiano
plt.axhline(0, c='black', lw=0.5)
plt.axvline(0, c='black', lw=0.5)
plt.xlim(-5,5)
plt.ylim(-5,5)

# Criamos os vetores
plt.quiver(0, 0, 2, 3, angles='xy', scale_units='xy', scale=1, color='blue')
plt.quiver(0, 0, -2, -3, angles='xy', scale_units='xy', scale=1, color='orange')
plt.quiver(0, 0, 4, 3, angles='xy', scale_units='xy', scale=1, color='red')
plt.quiver(0, 0, 1, 4, angles='xy', scale_units='xy', scale=1, color='green')

plt.show()
```

limites de x e y

todos começam com (0,0)

linhas dos vetores



Exercício 4

Como as linhas de uma matriz, os vetores podem ser adicionados e subtraídos. Quando adicionamos ou subtraímos vetores, elementos na mesma posição são adicionados ou subtraídos e um novo vetor é criado.

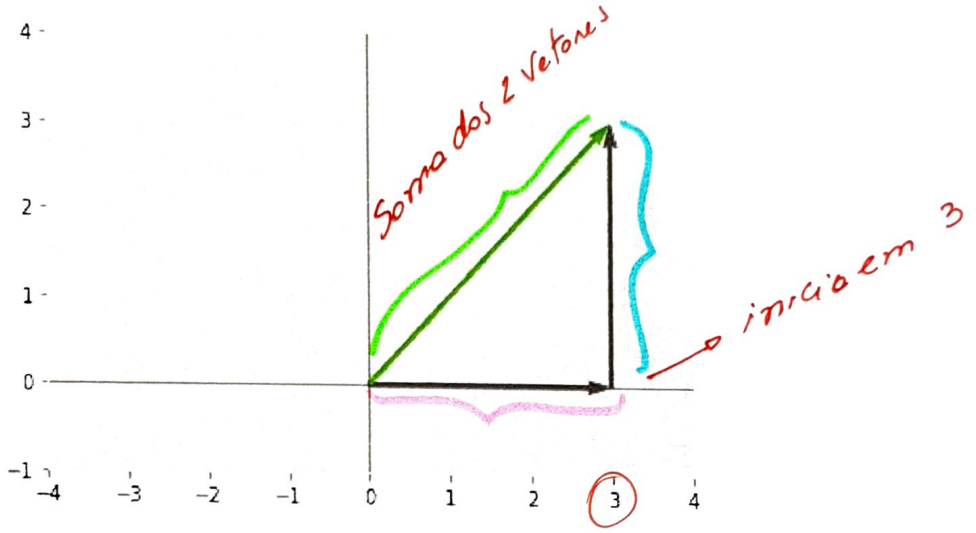
Usando a função pyplot.quiver(), crie o plot que mostre a adição dos vetores (3, 0) e (0, 3). Os dois vetores também devem ser mostrados no plot

In [4]:

```
# Solução

# Desenhamos o plano cartesiano
plt.axhline(0, c='black', lw=0.5)
plt.axvline(0, c='black', lw=0.5)
plt.xlim(-4,4)
plt.ylim(-1,4)

# Criamos os vetores
plt.quiver(0, 0, 3, 0, angles='xy', scale_units='xy', scale=1)
plt.quiver(3, 0, 0, 3, angles='xy', scale_units='xy', scale=1)
plt.quiver(0, 0, 3, 3, angles='xy', scale_units='xy', scale=1, color='green')
```

Out[4]:

```
<matplotlib.quiver.Quiver at 0x111287160>
```



Exercício 5

Podemos também dimensionar vetores multiplicando ou dividindo por um escalar (um número real). Entre adição vetorial, subtração vetorial, multiplicação escalar e divisão escalar, podemos combinar vetores de muitas maneiras diferentes. Ser capaz de combinar vetores para criar outros vetores é uma ideia-chave na álgebra linear.

Usando a função pyplot.quiver(), crie um plot que mostre:

- O vetor (3, 1)
- O vetor resultado da operação (3, 1) + 2(1, 0)
- O vetor resultado da operação 3(3, 1) + 2(-1, -1)

$3 \times 1 = 3$ $\quad 1 \times 1 = 1$ $\quad 2 \times 1 = 2$

$3 \times 0 = 0$ $\quad 1 \times 0 = 0$ $\quad 2 \times 0 = 0$

$3 \quad 1 \quad 2$

$2 \cdot (1,0)$

$2 \times 1 = 2$

$2 \times 0 = 0$

$(3,1) + (2,0)$

$3+2 = 5$

$1+0 = 1$

$3 \cdot (3,1) = 3 \times 3 = 9$

$3 \times 1 = 3$

$(9,3)$

$2 \cdot (-1,-1) = 2 \times -1 = -2$

$2 \times -1 = -2$

$(-2,-2)$

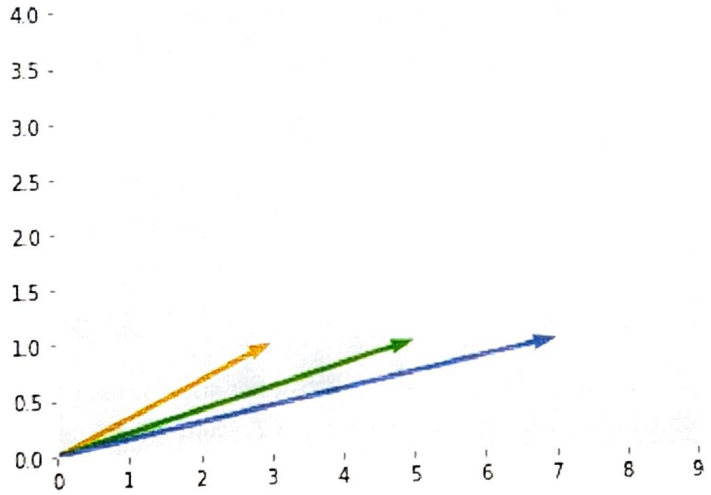
$(9-2)\ (3-2) = (7,1)$

In [5]:

```
# Solução

# Desenhamos o plano cartesiano
plt.axhline(0, c='black', lw=1)
plt.axvline(0, c='black', lw=1)
plt.xlim(0,9)
plt.ylim(0,4)

# Criamos os vetores
plt.quiver(0, 0, 3, 1, angles='xy', scale_units='xy', scale=1, color='orange')
plt.quiver(0, 0, 5, 1, angles='xy', scale_units='xy', scale=1, color='green')
plt.quiver(0, 0, 7, 1, angles='xy', scale_units='xy', scale=1, color='blue')
```

Out[5]:

```
<matplotlib.quiver.Quiver at 0x11134c128>
```



Exercício 6

Agora que você sabe trabalhar com os vetores de forma visual de como os vetores funcionam, vamos ver como podemos calcular usando vetores no NumPy.

Multiplique vetor v1 abaixo por 2, multiplique o vetor v2 por 5 e adicione ambos os vetores em uma combinação linear. Atribua o vetor resultante a v3.

In [6]:

```
# Solução

import numpy as np

v1 = np.asarray([
    [1],
    [2],
    [1]
], dtype=np.float32)

v2 = np.asarray([
    [3],
    [0],
    [1]
], dtype=np.float32)

v3 = (2 * v1) + (5 * v2)

print(v3)
```

```
[[17.]
 [ 4.]
 [ 7.]]
```

Exercício 7

Outra operação de vetores é o produto escalar (dot product). O produto escalar é semelhante à multiplicação, mas envolve a multiplicação dos elementos nos vetores de uma maneira específica. Ambos os vetores precisam conter o mesmo número de elementos. Para calcular o produto escalar, precisamos somar os produtos dos dois valores em cada posição em cada vetor.

Ao contrário das outras operações vetoriais, o resultado de um produto escalar é um valor escalar, não um vetor. Para calcular o produto escalar entre dois vetores, precisamos usar a função numpy.dot(). Esta função aceita 2 objetos ndarray como os 2 parâmetros requeridos. A principal peculiaridade é que um dos dois vetores precisa ser representado como um vetor de linha enquanto o outro vetor de coluna.

Calcule o dot product dos vetores v1 e v2 abaixo.

In [7]:

```
# Solução

import numpy as np

v1 = np.asarray([
    [1],
    [2],
    [1]
], dtype=np.float32)

v2 = np.asarray([
    [3],
    [0],
    [1]
], dtype=np.float32)

# Precisamos converter um dos vetores para um vetor linha
dot_product = np.dot(v1[:,0], v2)

# Se não fazemos a conversão de um dos vetores, teremos uma mensagem de erro (descomente e
# dot_product = np.dot(v1, v2)

print(dot_product)
```

converte o vetor v1 [1, 2, 1] mesmo resultado do transpose

```
[4.]
```

Exercícios 8, 9 e 10

Eu tenho três matrizes numpy:

- X: uma matriz de 3073 x 49000
- W: uma matriz de 10 x 3073
- y: um vetor de 49000 x 1

y contém valores entre 0 e 9, cada valor representa uma linha em W.

Eu gostaria de adicionar a primeira coluna de X à linha em W dada pelo primeiro elemento em y. Ou seja se o primeiro elemento em y for 3, adicione a primeira coluna de X à quarta linha de W. Em seguida, adicione a segunda coluna de X à linha em W fornecida pelo segundo elemento em y, e assim por diante, até que todas as colunas de X sejam adicionadas à linha em W especificada por y, o que significa um total de 49.000 linhas adicionadas.

W [y] + = X.T não resolve, porque isso não adicionará mais de um vetor a uma linha em W.

Você consegue encontrar uma solução? (para reduzir o tempo de processamento e facilitar a demonstração da solução, usaremos matrizes com dimensões menores). Crie uma solução:

1. Com loop
2. Usando a função add.at() do NumPy
3. Solução vetorizada com a função einsum() do NumPy

Nota: Estou procurando por soluções vetorizadas. Ou seja não há loops. Esse seria o resultado esperado:

W = [[17, 22, 16, 16], [7, 11, 14, 17], [8, 2, 10, 11]]

Em matemática, especialmente em álgebra linear e teoria de matriz, a vetorização de uma matriz é uma transformação linear que converte a matriz em um vetor de coluna. Especificamente, a vetorização de uma matriz m × n A, indicada por vec(A), é o vetor da coluna mn × 1 obtido empilhando as colunas da matriz A uma sobre a outra.

In [8]:

```
# Solução com Loop

import numpy as np

a, b, c = 3, 4, 5

np.random.seed(0)

X = np.random.randint(10, size=(b,c))
W = np.random.randint(10, size=(a,b))
y = np.random.randint(a, size=(c,1))

for i,j in enumerate(y):
    W[j]+=X[:,i]

print(W)
```

```
[[17 22 16 16]
 [ 7 11 14 17]
 [ 8  2 10 11]]
```

t = ['a', 'b', 'c']
for i, word in enumerate(t)
Print (i, word)
0 a
1 b
2 c

4 5

Valor de y

5

Nº
0
1
2
3

Solução com loop é ineficiente
o pessoa que propõe não tem
tanto conhecimento

Dica entrevista

A função add.at() faz um cálculo sem buffer, contornando o buffer que impede que o W [y.ravel ()] + = X.T funcione. Ainda é iterativo, mas o loop foi movido para o código compilado. Não é vetorização verdadeira porque a ordem de aplicação é importante. A adição de uma linha de X.T depende dos resultados das linhas anteriores.

In [9]:

```
# Solução com add.at()

import numpy as np

a, b, c = 3, 4, 5

np.random.seed(0)

X = np.random.randint(10, size=(b,c))
W = np.random.randint(10, size=(a,b))
y = np.random.randint(a, size=(c,1))

np.add.at(W,(y.ravel()),X.T)
print(W)
```

```
[[17 22 16 16]
 [ 7 11 14 17]
 [ 8  2 10 11]]
```

x = np.array([[1,2,3],[4,5,6]])
Print(np.ravel(x))
[1,2,3,4,5,6]

0 1 2 3
a = np.array([1,2,3,4])
np.add.at(a,[1,2,3], 3)
Print(a)
[1,5,6,7]

Somo o Nº 3
Nas posições 1,2,3
2,3,4
5,6,7

O que faz o einsum? Imagine que temos duas matrizes multidimensionais, A e B. Agora vamos supor que queremos ...

- Multiplicar A por B de uma maneira particular para criar uma nova matriz
- Somar esta nova matriz ao longo de eixos específicos
- Transpor os eixos da nova matriz em uma ordem específica.

Há uma boa chance de que o einsum nos ajude a fazer isso mais rápido e com mais eficiência de memória que as combinações das funções NumPy, como multiplicação, soma e transposição, permitirão.

Veja mais exemplos desta função neste excelente post:
https://stackoverflow.com/questions/26089893/understanding-numpys-einsum
(https://stackoverflow.com/questions/26089893/understanding-numpys-einsum)

Vamos colocar um exemplo que deve facilitar a compreensão:

Imagine que queremos multiplicar as matrizes A e B element-wise e, em seguida, somar ao longo das linhas da nova matriz. No NumPy "normal" nós escreveríamos algo assim:

In [10]:

```
import numpy as np

A = np.array([0, 1, 2])

B = np.array([[ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11]])

(A[:, np.newaxis] * B).sum(axis=1)
```

soma

→ x 0 = 0,0,0,0 = 0
— x 1 = 4,5,6,7 = 22
→ x 2 = 16,18,20,22 = 76

soma de linhas

Out[10]:

```
array([ 0, 22, 76])
```

A operação de indexação em A alinha os primeiros eixos das duas matrizes para que a multiplicação possa ser feita. As linhas da matriz de produtos são então somadas para retornar a resposta.

Agora, se quiséssemos usar o einsum, poderíamos escrever:

ordem em que será feito os cálculos

In [11]:

```
np.einsum('i,ij->i', A, B)
```

Out[11]:

```
array([ 0, 22, 76])
```

A string 'i, ij-> i' é o que determina o funcionamento da função. Você pode pensar nisso em duas metades. No lado esquerdo (à esquerda do ->) rotulamos os dois arrays de entrada. À direita de ->, rotulamos o array com o qual queremos o resultado final.

Aqui está o que acontece a seguir:

A tem um eixo e nós rotulamos isso como i. E B tem dois eixos; rotulamos o eixo 0 como i e o eixo 1 como j.

Ao repetir o rótulo i em ambas as matrizes de entrada, estamos dizendo que esses dois eixos devem ser multiplicados juntos. Em outras palavras, estamos multiplicando o array A com cada coluna do array B, assim como A [:, np.newaxis] * B faz.

Observe que j não aparece como um rótulo em nossa saída desejada; Acabamos de usar o i (queremos acabar com um array 1D). Ao omitir o rótulo, estamos informando o einsum para somar ao longo deste eixo. Em outras palavras, estamos somando as linhas dos produtos, assim como o .sum (eixo = 1).

Isso é basicamente tudo que você precisa saber para usar o einsum. Se deixarmos os dois rótulos na saída, 'i, ij-> ij', receberemos uma matriz 2D de produtos (o mesmo que A [:, np.newaxis] *B). Se dissermos que não há rótulos de saída, 'i, ij->, retornamos um único número (o mesmo que fazer (A [:, np.newaxis]* B) .sum ()).

Quando trabalhamos com mais dimensões, usamos outras indexadores, como k e l.

In [12]:

```
# Solução vetorizada
# https://docs.scipy.org/doc/numpy/reference/generated/numpy.einsum.html

import numpy as np

a, b, c = 3, 4, 5

np.random.seed(0)

X = np.random.randint(10, size=(b,c))
W = np.random.randint(10, size=(a,b))
y = np.random.randint(a, size=(c,1))

N = y.max()+1

# Aqui, primeiro modificamos a dimensão de y com a função ravel() e rearranjamos o vetor y
# com base nos critérios da função np.einsum(), ou seja, 'ijk,lk->il'. Por fim, fazemos a c
# gravamos na matriz W em todas as linhas com número de colunas igual a N.
# Consulte o help da função np.einsum() para mais detalhes.
W[:N] += np.einsum('ijk,lk->il',(np.arange(N)[:,None,None] == y.ravel()),X)
print(W)
```

```
[[17 22 16 16]
 [ 7 11 14 17]
 [ 8  2 10 11]]
```

Vetorização é a melhor Resposta

In [13]:

```
?np.einsum
```

Help do Jupyter

Fim

Eigendecomposition de uma matriz é um tipo de decomposição que envolve a decomposição de uma matriz quadrada em um conjunto de autovetores e autovalores.

Um vetor é um autovetor de uma matriz se satisfizer a seguinte equação:

A . v = lambda . v

Ou sem a notação de ponto, apenas:

Av = lambdav

Isso é chamado de equação de autovalor, onde A é a matriz quadrada que estamos decompondo, v é o autovetor da matriz e lambda é a letra grega em minúscula e representa o escalar de autovalor.

Uma matriz pode ter um autovetor e um autovalor para cada dimensão da matriz pai. Nem todas as matrizes quadradas podem ser decompostas em autovetores e autovalores, e algumas só podem ser decompostas de uma forma que requer números complexos. A matriz pai pode ser mostrada como sendo um produto dos autovetores e autovalores.

A = Q . diag(V) . Q^-1

Onde Q é uma matriz formada pelos autovetores, diag(V) é uma matriz diagonal composta pelos autovalores ao longo da diagonal e Q ^ -1 é o inverso da matriz composta pelos autovetores. Esta equação também pode ser representada assim:

$$A = Q\Lambda Q^{-1}$$

Uma operação de decomposição não resulta em uma compactação da matriz; em vez disso, divide-se em partes constituintes para facilitar certas operações na matriz. Como outros métodos de decomposição de matrizes, a Eigendecomposition é usada como um elemento para simplificar o cálculo de outras operações de matriz mais complexas.

Eigendecomposition também pode ser usado para calcular os componentes principais de uma matriz no método de Análise de Componente Principal ou PCA que pode ser usado para reduzir a dimensionalidade dos dados em Machine Learning.

Definiremos agora o que significa polinômio de uma matriz e, a seguir, um teorema que relaciona uma matriz com um polinômio. Seja:

$$P(t) \;=\; a_0\, t^n + a_1\, t^{n-1} + \ldots + a_{n-1}\, t + a_n$$

um polinômio de grau n, onde os ai, i = 1, 2,...,n, são reais. Se A é uma matriz quadrada real, então definimos:

$$P(A) \;=\; a_0\, A^n + a_1\, A^{n-1} + \ldots + a_{n-1}\, A + a_n\, I$$

como sendo o polinômio da matriz A. Na expressão, I é a matriz identidade. Em particular, se P(A)=Θ (matriz nula), dizemos que A é um zero (ou raiz) de P(t).

De acordo com o Teorema de Cayley-Hamilton: Toda matriz é um zero do seu polinômio característico, isto é, se:

$$P(\lambda) = a_0\lambda^n + a_1\lambda^{n-1} + \ldots + a_n$$

é o polinômio característico de A, então P(A)=Θ.

Referências:

Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares
por Stephen Boyd (Autor), Lieven Vandenberghe (Autor)

Vejamos algumas propriedades que os autovalores satisfazem. Consideremos que A seja uma matriz de ordem n e que k seja um escalar qualquer. Então, são válidas as seguintes propriedades:

- Propriedade 1: Se λ é um autovalor de A, então λ é autovalor da transposta de A.
- Propriedade 2: Se λ é um autovalor de A, então $k\lambda$ é autovalor de kA.
- Propriedade 3: Se λ é um autovalor de A, então λ elevado a k é autovalor de A elevado a k.
- Propriedade 4: Se λ é um autovalor de A, e A é não singular, então λ elevado a -1 é autovalor de A elevado a -1.
- Propriedade 5: Se λ é um autovalor de A, então $\lambda + k$ é autovalor de $A + kI$, onde I é a matriz identidade.
- Propriedade 6: $\lambda = 0$ é autovalor de A se e somente se A é singular.

Referências:

Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares
por Stephen Boyd (Autor), Lieven Vandenberghe (Autor)

Vamos formalizar agora os conceitos a respeito da determinação dos autovalores e autovetores de matrizes. Dada uma matriz quadrada A, n × n, a matriz:

$$A - \lambda I = \begin{pmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{pmatrix}$$

onde I é a matriz identidade de ordem n e λ é um parâmetro, é chamada matriz característica de A. Seu determinante, det(A − λI), é um polinômio de grau n em λ, indicado por P(λ), e é chamado **polinômio característico** de A. A equação P(λ)=0 é chamada equação característica de A.

Para calcular o polinômio característico de A, basta calcular o determinante de A − λI. Por exemplo, o polinômio característico da matriz abaixo, seria:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$det(A - \lambda I) = \begin{vmatrix} 1 - \lambda & 2 \\ 3 & 4 - \lambda \end{vmatrix} = \underbrace{\lambda^2 - 5\lambda - 2}_{\text{polinômio característico}}.$$

Com isso, podemos deduzir alguns teoremas:

1- Um escalar λ é um autovalor de A se e somente se λ é um zero ou raiz do seu polinômio característico.

2- Existe no mínimo um autovetor correspondente a cada autovalor.

3- Os autovetores correspondentes a cada autovalor formam um espaço vetorial (o que chamamos de autoespaço).

4- Os autovetores correspondentes a autovalores distintos são LI.

5- Se A tem n autovalores distintos, então existem exatamente n autovetores; um associado a cada autovalor.

6- Se A é uma matriz de ordem n triangular superior (ou inferior), então os autovalores de A são seus elementos diagonais.

Referências:

Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares
por Stephen Boyd (Autor), Lieven Vandenberghe (Autor)

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 7

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Exercício 1 - Resolva o sistema de equações:

1x + 1y = 18

2x + 4y = 24

In [1]:

```
# Solução

import numpy as np
expressao1 = np.array([[1, 1],[2,4]])
expressao2 = np.array([18, 24])

x, y = np.linalg.solve(expressao1, expressao2)
print("Os valores de x e y nas equações acima são, respectivamente:", x, y)
```

matriz de coeficientes

vetor de termos independentes

```
Os valores de x e y nas equações acima são, respectivamente: 24.0 -6.0
```

Exercício 2 - Resolva o sistema linear abaixo:

x+y+z = 2

2x-6y-z=-1

linha

3x-2z=8

Dica: organize os coeficientes na orientação das linhas no sistema linear.

In [2]:

```
# Solução

import numpy as np
a = np.array([[1,1,1],[2,-6,-1],[3,0,-2]])
b = np.array([2,-1,8])

resultado1 = np.linalg.solve(a, b)
print(resultado1)
```

x y z

(-z)

(y)

Não opõe y, logo y = 0

orientação por linha

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & -6 & -1 \\ 3 & 0 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 8 \end{bmatrix}$$

```
[ 2.  1. -1.]
```

Exercício 3 - Resolva o sistema linear abaixo:

coluna

x+y+z = 2

2x-6y-z=-1

3x-2z=8

Dica: organize os coeficientes na orientação das colunas no sistema linear.

In [3]:

```
# Solução

import numpy as np
a = np.array([[1,2,3],[1,-6,0],[1,-1,-2]])
b = np.array([2,-1,8])

resultado2 = np.linalg.solve(a.T, b)
print(resultado2)
```

orientação por coluna

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & -6 & 0 \\ 1 & -1 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 8 \end{bmatrix}$$

transposto da matriz

```
[ 2.  1. -1.]
```

Exercício 4 - Resolva o sistema linear abaixo usando o método da matriz inversa:

x+y+z = 2

2x-6y-z=-1

3x-2z=8

In [4]:

```
# Solução

import numpy as np
a = np.array([[1,2,3],[1,-6,0],[1,-1,-2]])
b = np.array([2,-1,8])

# X = (A-1) . b
resultado3 = np.dot(b, np.linalg.inv(a))
print(resultado3)
```

→ termos independentes

↳ inversa da matriz(a)

determinante

$$\begin{array}{ccccc} 1 & 2 & 3 & 1 & 2 \\ 1 & -6 & 0 & 1 & -6 \\ 1 & -1 & -2 & 1 & -6 \end{array}$$

$$\begin{array}{ccccc} - & - & - & + & + & + \end{array}$$

$-18+0-4+12+0-18$

$-22+12-18$

$-10-18$

-28

```
[ 2.  1. -1.]
```

Exercício 5 - Resolva o sistema linear abaixo:

3x+2y-4z+t = 30

2x+3y+3z+2t=150

5x-3y+z-5t=140

x+y+z+t=60

$$\begin{bmatrix} 3 & 2 & -4 & 1 \\ 2 & 3 & 3 & 2 \\ 5 & -3 & 1 & -5 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 30 \\ 150 \\ 140 \\ 60 \end{bmatrix}$$

In [5]:

```
# Solução

A = np.array([[3, 2, -4, 1], [2, 3, 3, 2], [5, -3, 1, -5], [1, 1, 1, 1]])
b = np.array([30, 150, 140, 60])
resultado1 = np.linalg.solve(A, b)
print(resultado1)
```

```
[3.00000000e+01 1.00000000e+01 2.00000000e+01 2.77734574e-15]
```

Exercício 6 (Desafio) - Desenvolva uma função em Python que obtenha o mesmo resultado da função solve() abaixo.

Dica: sua função deve receber a equação como parâmetro e não as matrizes.

In [6]:

```
import numpy as np
a = np.array([[3,5],[12,-3]])
b = np.array([[29],[24]])
np.linalg.solve(a,b)
```

Out[6]:

```
array([[3.],
       [4.]])
```

In [7]:

Recebe equação

variáveis x e y

```
# Solução

def mysolve(eq, var=('x', 'y')):
    import re
```

Regular expression

```
    var_re = re.compile(r'(\+|\-)\s*(\d*)\s*\*?\s*(x|y)')
    const_re = re.compile(r'(\+|\-)\s*(\-?\d+)$')
```

Consegue extrair de uma string Padrões de caracteres

```
    constants, eqns, coeffs, default = [],[], {'x': [], 'y': []}, {'': '1'}

    for e in eq.split(';'):
        eq1 = e.replace("=", " - ").strip()
        if not eq1.startswith('-'):
            eq1 = '+' + eq1
        eqns.append(eq1)
```

divide no ;

```
    var_eq1, var_eq2 = map(var_re.findall, eqns)

    constants = [-1*int(x[0][1]) for x in map(const_re.findall, eqns)]
    [coeffs[x[2]].append(int((x[0]+ default.get(x[1], x[1])).strip())) for x in (var_eq1 +
                                                                                var_eq2)]

    ycoeff = coeffs['y']
    xcoeff = coeffs['x']

    # Ajuste as equações para receber y e resolva para x
    if ycoeff[0]*ycoeff[1] > 0:
        ycoeff[1] *= -1
        xcoeff[0] *= ycoeff[1]
        constants[0] *= -1*ycoeff[1]
    else:
        xcoeff[0] *= -1*ycoeff[1]
        constants[0] *= ycoeff[1]
```

equivalente a função Solve

```
    xcoeff[1] *= ycoeff[0]
    constants[1] *= -1*ycoeff[0]

    # Obtém x
    xval = sum(constants)*1.0/sum(xcoeff)

    # Agora resolva para y usando o valor de x
    z = eval(eqns[0],{'x': xval, 'y': 1j})
    yval = -z.real*1.0/z.imag

    return (xval, yval)
```

In [8]:

```
# Solução

mysolve('3*x + 5*y = 29; 12*x - 3*y = 24')
```

Out[8]:

```
(3.0, 4.0)
```

Fim

```python
# Reconhecimento Facial com Eigenfaces

# Execute o comando abaixo no terminal ou prompt de comando:
# python eigenfaces_face_recognition.py --dataset caltech_faces

# Dataset: http://www.vision.caltech.edu/html-files/archive.html

# O dataset CALTECH Faces é um conjunto de dados de referência popular para algoritmos de reconhecimento facial.
# No geral, o conjunto de dados é composto por 450 imagens de aproximadamente 27 pessoas únicas.
# Cada foto foi capturada em várias condições de iluminação, cenas de fundo e expressões faciais.

# Referências:
# https://www.lcg.ufrj.br/marroquim/courses/cos756/trabalhos/2013/abel-nascimento/abel-nascimento-report.pdf
# https://en.wikipedia.org/wiki/Eigenface
# https://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf
# https://arxiv.org/pdf/1705.02782.pdf
# http://vision.stanford.edu/teaching/cs231a_autumn1112/lecture/lecture2_face_recognition_cs231a_marked.pdf


# Imports
# pip install imutils
# pip install opencv-python
import numpy as np

import argparse

import imutils

import cv2 → opencv

from datasets import load_caltech_faces  Função que nos permite carregar o dataset

from resultsmontage import ResultsMontage → Gerar os componentes Principais que está no outro script

from sklearn.decomposition import PCA

from sklearn.metrics import classification_report

from sklearn.svm import SVC → classificador do suporte vector machine

from skimage import exposure → mostra a imagem na tela

# Argumentos

ap = argparse.ArgumentParser()  leitura dos argumentos

ap.add_argument("-d", "--dataset", required=True, help="Caminho para o dataset da Caltech")

ap.add_argument("-n", "--num-components", type=int, default=150, help="Número de Componentes Principais")

ap.add_argument("-s", "--sample-size", type=int, default=10, help="Número de Amostras")

ap.add_argument("-v", "--visualize", type=int, default=-1, help="Se os componentes PCA devem ser visualizados")

args = vars(ap.parse_args())

# Carrega o dataset de faces
```

```python
print("Carregando o Dataset da Caltech...")

# Não fizemos nenhum tratamento especial nas imagens.
# Apenas separamos as fotos de 21 em 21 (21 fotos de cada pessoa), para poder usar isso como parâmetro e assim customizar o treinamento e reduzir o tempo total.
# O parâmetro min_faces na chamada a função load_caltech_faces() controla exatamente isso. O parâmetro está com o valor 21.
(training, testing, names) = load_caltech_faces(args["dataset"], min_faces=21, flatten=True, test_size=0.25)

# Computa a representação PCA (eigenfaces) dos dados e, em seguida, projeta os dados de treinamento no subespaço de autodefinições
print("Criando Eigenfaces...")

pca = PCA(n_components=args["num_components"], whiten=True)

trainData = pca.fit_transform(training.data)

# Verifica se os componentes PCA devem ser visualizados
if args["visualize"] > 0:
    # Inicializa a montagem para os componentes
    montage = ResultsMontage((62, 47), 4, 16)

    # Loop sobre os primeiros 16 componentes individuais
    for (i, component) in enumerate(pca.components_[:16]):

        # Reshape do componente para uma matriz 2D e, em seguida, converte o tipo de dados em um inteiro de 8 bits
        # para que ele possa ser exibido com o OpenCV
        component = component.reshape((62, 47))

        component = exposure.rescale_intensity(component, out_range=(0, 255)).astype("uint8")

        component = np.dstack([component] * 3)

        montage.addResult(component)

    # Mostra a média e as visualizações dos componentes principais
    # Mostra a imagem média
    mean = pca.mean_.reshape((62, 47))

    mean = exposure.rescale_intensity(mean, out_range=(0, 255)).astype("uint8")

    cv2.imshow("Média", mean)

    cv2.imshow("Componentes", montage.montage)

    cv2.waitKey(0)

# Treina um classificador na representação de eigenfaces
print("Treinando o classificador...")

model = SVC(kernel="rbf", C=10.0, gamma=0.001, random_state=84)

model.fit(trainData, training.target)

# Avalia o modelo
print("Avaliando o modelo...")

predictions = model.predict(pca.transform(testing.data))

print(classification_report(testing.target, predictions))

# Loop sobre o número desejado de amostras

for i in np.random.randint(0, high=len(testing.data), size=(args["sample_size"],)):
```

arquivo datasets.py

Converte matriz em vetor

Recebe o dataset

no mínimo de faces

ISO

Cria o objeto pca mumuscado a partir do Classe PCA

extrai os autovalores e autovetores

Matriz de covariância

dados de entrada

dados de saída

Fit = treinamento

algoritmo do scikitlearning

Preciso passar a imagem no mesmo formato

Passando imagens aleatórias

```
# Obtém o rosto e classifica
face = testing.data[i].reshape((62, 47)).astype("uint8")

prediction = model.predict(pca.transform(testing.data[i].reshape(1, -1)))

# Redimensiona o rosto para torná-lo mais visível, então exibe o rosto e a previsão
print("Previsão: {}, Atual: {}".format(prediction[0], testing.target[i]))

face = imutils.resize(face, width=face.shape[1] * 2, inter=cv2.INTER_CUBIC)

cv2.imshow("Face", face)

cv2.waitKey(0)
```

```python
# Módulo para Carregar o Dataset

# Imports
from sklearn.datasets.base import Bunch
from imutils import paths
from scipy import io
import numpy as np
import random
import cv2

def load_caltech_faces(datasetPath, min_faces=10, face_size=(47, 62), equal_samples=True, test_size=0.33, seed=42, flatten=False):

    # Obtém os caminhos da imagem associados às faces e carrega os dados da caixa delimitadora

    imagePaths = sorted(list(paths.list_images(datasetPath)))

    bbData = io.loadmat("{}/ImageData.mat".format(datasetPath))

    bbData = bbData["SubDir_Data"].T

    # Define a semente aleatória, inicializa a matriz de dados e os rótulos
    random.seed(seed)

    data = []

    labels = []

    # Loop pelas imagens
    for (i, imagePath) in enumerate(imagePaths):

        # Carrega a imagem e converte para Grayscale
        image = cv2.imread(imagePath)

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Obtém a caixa delimitadora associada à imagem atual, extraia o ROI do rosto e redimensione-o para um tamanho canônico
        k = int(imagePath[imagePath.rfind("_") + 1:][:4]) - 1

        (xBL, yBL, xTL, yTL, xTR, yTR, xBR, yBR) = bbData[k].astype("int")

        face = gray[yTL:yBR, xTL:xBR]

        face = cv2.resize(face, face_size)

        # Verifica se o rosto deve ser "achatado" (flatten) em uma única linha
        if flatten:
            face = face.flatten()

        # Atualiza a matriz de dados e as etiquetas associadas
        data.append(face)

        labels.append(imagePath.split("/")[-2])

    # Converte a matriz de dados e a lista de etiquetas para uma matriz NumPy
    data = np.array(data)

    labels = np.array(labels)

    # Verifica se as amostras iguais para cada face devem ser usadas
    if equal_samples:
        # Inicializa a lista de índices

        sampledIdxs = []

        # Loop pelos labels únicos
```

Faz a leitura da imagem

lê a imagem

Converte p/ escala de cinza

tendo apenas o canal preto e branco fica mais fácil unidimensional

extrai as coordenadas dos dados

Matriz

etiquetas o nome do diretório como label

deu erro provavelmente porque ele usava mac

```
        for label in np.unique(labels):
            # Obtém os índices na matriz de etiquetas onde os rótulos são iguais ao rótulo atual

            labelIdxs = np.where(labels == label)[0]

            # Prossegue apenas se o número necessário de rostos mínimos puderem ser atendidos
            if len(labelIdxs) >= min_faces:
                # Aleatoriamente gera os índices para o rótulo atual, mantendo apenas o valor mínimo fornecido, e atualiza a lista de índices amostrados

                labelIdxs = random.sample(list(labelIdxs), min_faces)

                sampledIdxs.extend(labelIdxs)

        # Usa os índices amostrados para selecionar os pontos e os rótulos de dados apropriados
        random.shuffle(sampledIdxs)

        data = data[sampledIdxs]

        labels = labels[sampledIdxs]

    # Calcula o índice de divisão de treinamento e teste
    idxs = list(range(0, len(data)))

    random.shuffle(idxs)

    split = int(len(idxs) * (1.0 - test_size))

    # Divide os dados em segmentos de treinamento e teste
    (trainData, testData) = (data[:split], data[split:])

    (trainLabels, testLabels) = (labels[:split], labels[split:])

    # Cria os grupos de treinamento e teste
    training = Bunch(name="training", data=trainData, target=trainLabels)

    testing = Bunch(name="testing", data=testData, target=testLabels)

    # Devolve uma tupla do treinamento
    return (training, testing, labels)
```

embaralhar e dispor tudo de forma aleatória

Preparação dos dados

```python
# Imprime os componentes

# Imports
import numpy as np

import cv2

class ResultsMontage:
    def __init__(self, imageSize, imagesPerRow, numResults):

        # Armazena o tamanho da imagem alvo e o número de imagens por linha

        self.imageW = imageSize[0]

        self.imageH = imageSize[1]

        self.imagesPerRow = imagesPerRow

        # Aloca memória para a imagem de saída

        numCols = numResults // imagesPerRow

        self.montage = np.zeros((numCols * self.imageW, imagesPerRow * self.imageH, 3), dtype="uint8")

        # Inicializa o contador para a imagem atual juntamente com o número da linha e da coluna

        self.counter = 0

        self.row = 0

        self.col = 0

    def addResult(self, image, text=None, highlight=False):
        # Verifica se o número de imagens por linha foi obtido e, em caso afirmativo, redefine o contador de colunas e incrementa a linha

        if self.counter != 0 and self.counter % self.imagesPerRow == 0:

            self.col = 0

            self.row += 1

        # Redimensiona a imagem para a largura e a altura fixas e configura na montagem
        image = cv2.resize(image, (self.imageH, self.imageW))

        (startY, endY) = (self.row * self.imageW, (self.row + 1) * self.imageW)

        (startX, endX) = (self.col * self.imageH, (self.col + 1) * self.imageH)

        self.montage[startY:endY, startX:endX] = image

        # Se o texto não for None, desenhe
        if text is not None:

            cv2.putText(self.montage, text, (startX + 10, startY + 30), cv2.FONT_HERSHEY_SIMPLEX,

                1.0, (0, 255, 255), 3)

        # Verifique se o resultado deve ser destacado
        if highlight:

            cv2.rectangle(self.montage, (startX + 3, startY + 3), (endX - 3, endY - 3), (0, 255, 0), 4)

```

```
64        # Incrementa o contador de colunas e o contador de imagens
65        self.col += 1
66
67        self.counter += 1
```

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 8

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Exercício 1 - Usando a função numpy.linalg.eigvals() do Numpy, calcule os autovalores da matriz abaixo.

In [1]:

```
# Solução
import numpy as np

m = np.matrix([[2, -1, 0],
               [-1, 2, -1],
               [0, -1, 2]])

eigenvalues = np.linalg.eigvals(m)

print("\nAutovalores: ", eigenvalues)
```

```
Autovalores:  [3.41421356 2.         0.58578644]
```

Exercício 2 - Usando a função numpy.linalg.eig() do Numpy, calcule os autovalores e autovetores da matriz abaixo.

Dica: A função numpy.linalg.eig() retorna uma tupla que consiste em um vetor e uma matriz. O vetor (que pode ser representado por "autova") contém os autovalores. O array (que pode ser representado por "autove") contém os autovetores correspondentes, um autovetor por coluna. Os autovetores são normalizados para que suas normas euclidianas sejam 1.

Referência: https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.eig.html (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.eig.html)

In [2]:

```
# Solução
import numpy as np

from numpy import linalg as LA

A = np.array([[1,2,3],[3,2,1],[1,0,-1]])

autova, autove = LA.eig(A)

print("\nAutovalores: ", autova)

print("\nAutovetores: ", autove)
```

Retorna dois conjuntos de valores

```
Autovalores:  [ 4.31662479e+00 -2.31662479e+00  1.47314580e-16]

Autovetores:  [[ 0.58428153  0.73595785  0.40824829]
 [ 0.80407569 -0.38198836 -0.81649658]
 [ 0.10989708 -0.55897311  0.40824829]]
```

Exercício 3 - Usando as funções oferecidas pelo SymPy, calcule autovalores e autovetores da matriz abaixo.

In [3]:

```
# Solução
from sympy import *

A = Matrix([[0, 2], [1, -3]])

print("\nAutovalores: ", A.eigenvals())   # retorna autovalores e sua multiplicidade algébr
print("\nAutovetores: ", A.eigenvects())  # retorna autovalores, autovetores
```

```
Autovalores:  {-sqrt(17)/2 - 3/2: 1, -3/2 + sqrt(17)/2: 1}

Autovetores:  [(-3/2 + sqrt(17)/2, 1, [Matrix([
[-2/(-sqrt(17)/2 + 3/2)],
[                      1]])]), (-sqrt(17)/2 - 3/2, 1, [Matrix([
[-2/(3/2 + sqrt(17)/2)],
[                     1]])])]
```

Exercício 4 - Com base no exercício anterior, retorne apenas a lista de autovalores.

A.eigenvals () retorna {-sqrt (17) / 2 - 3/2: 1, -3/2 + sqrt (17) / 2: 1} que é um dicionário de autovalores e suas multiplicidades. Se você não precisa das multiplicidades, use list(A.eigenvals().Keys()) para obter uma lista simples de autovalores.

A saída de autovetores é um pouco mais complicada e consiste em triplos (autovalor, multiplicidade desse autovalor, base do autoespaço). Note que a multiplicidade é multiplicidade algébrica, enquanto o número de autovetores retornados é a multiplicidade geométrica, que pode ser menor. Os autovetores são retornados como matrizes de 1 coluna.

Para essa matriz, A.eigenvects() retorna o autovetor [-2 / (- sqrt (17) / 2 + 3/2), 1] para o autovalor -3/2 + sqrt (17) / 2, e autovetor [-2 / (3/2 + sqrt (17) / 2), 1] para eigenvalue -sqrt (17) / 2 - 3/2.

Se você quiser os autovetores apresentados como listas simples de coordenadas, usamos list comprehension:

[list (tup [2] [0]) fro tup em A.eigenvects ()]

que geraria [[-2 / (- sqrt (17) / 2 + 3/2), 1], [-2 / (3/2 + sqrt (17) / 2), 1]]. (Note que apenas escolhe um autovetor para cada autovalor, o que nem sempre é o que você quer).

In [4]:

```
# Solução
from sympy import *

A = Matrix([[0, 2], [1, -3]])

print("\nAutovalores: ", list(A.eigenvals().keys()))

print("\nAutovetores: ", [list(tup[2][0]) for tup in A.eigenvects()])
```

usamos list comprehension

Para extrair apenas a informação relevante, pois o eigenvects retorna muito mais inform.

```
Autovalores:  [-sqrt(17)/2 - 3/2, -3/2 + sqrt(17)/2]

Autovetores:  [[-2/(-sqrt(17)/2 + 3/2), 1], [-2/(3/2 + sqrt(17)/2), 1]]
```

Exercício 5 - Calcule o Eigendecomposition da matriz abaixo.

Uma eigendecomposition é calculada em uma matriz quadrada usando um algoritmo iterativo eficiente através de funções do NumPy.

Frequentemente, um autovalor é encontrado primeiro, então um autovetor é encontrado para resolver a equação como um conjunto de coeficientes.

A eigendecomposition pode ser calculada no NumPy usando a função eig(). A eigendecomposition é calculada na matriz retornando os autovalores e autovetores.

In [5]:

```
# Solução
from numpy import array

from numpy.linalg import eig

# Define a matriz

A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(A)

# Calcula eigendecomposition

values, vectors = eig(A)

# Print
print("\nAutovalores: ", values)

print("\nAutovetores: ",vectors)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Autovalores:  [ 1.61168440e+01 -1.11684397e+00 -9.75918483e-16]

Autovetores:  [[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

Exercício 6 - Confirme que um vetor é de fato um autovetor de uma matriz.

Fazemos isso multiplicando o autovetor candidato pelo autovetor e comparando o resultado com o autovalor.

Primeiro, definiremos uma matriz e calcularemos os autovalores e os autovetores. Vamos então testar se o primeiro vetor e valor são de fato um autovalor e um autovetor para a matriz. Nós sabemos que eles são, mas é um bom exercício.

Os autovetores são retornados como uma matriz com as mesmas dimensões que a matriz pai, onde cada coluna é um autovetor, isto é, o primeiro autovetor é o vetor [:, 0]. Os autovalores são retornados como uma lista, onde os índices de valor na matriz retornada são pareados com autovetores pelo índice da coluna, por exemplo. o primeiro autovalor em valores [0] é emparelhado com o primeiro autovetor em vetores [:, 0].

O exemplo multiplica a matriz original pelo primeiro autovetor e a compara ao primeiro autovetor multiplicado pelo primeiro autovalor.

Executar o exemplo imprime os resultados dessas duas multiplicações que mostram o mesmo vetor resultante, como seria de se esperar.

In [6]:

```
# Solução
from numpy import array
from numpy.linalg import eig

# Define a matriz

A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Calcula eigendecomposition

values, vectors = eig(A)

# Confirmando o primeiro eigenvector
B = A.dot(vectors[:, 0])

print(B)

C = vectors[:, 0] * values[0]

print(C)
```

dot do Matriz com seu auto vetor

todos as linhas da coluna zero

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

1, 4, 7

```
[ -3.73863537  -8.46653421 -13.19443305]
[ -3.73863537  -8.46653421 -13.19443305]
```

Exercício 7 - Com base nos autovalores e autovetores, reconstrua a matriz original.

Podemos inverter o processo e reconstruir a matriz original, recebendo apenas os autovetores e autovalores.

Primeiro, a lista de autovetores deve ser convertida em uma matriz, onde cada vetor se torna uma linha. Os autovalores precisam ser organizados em uma matriz diagonal. A função NumPy diag() pode ser usada para isso.

Em seguida, precisamos calcular o inverso da matriz do vetor próprio, o que podemos alcançar com a função invPP (NumPy). Finalmente, esses elementos precisam ser multiplicados junto com a função dot().

O exemplo calcula os autovalores e os autovetores novamente e os utiliza para reconstruir a matriz original.

A execução do exemplo primeiro imprime a matriz original e, em seguida, a matriz reconstruída a partir de autovalores e autovetores correspondentes à matriz original.

In [7]:

```
# Solução

from numpy import diag
from numpy import dot
from numpy.linalg import inv
from numpy import array
from numpy.linalg import eig

# Define a Matriz
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("\nMatriz Original:\n", A)

# Calcula autovalofes e autovetores
values, vectors = eig(A)

# Cria a matriz de autovetores
Q = vectors

# Cria a inversa da matriz de autovetores
R = inv(Q)

# Cria a matriz diagonal dos autovalores
L = diag(values)

# Reconstrói a matriz original
B = Q.dot(L).dot(R)

print("\nMatriz Original Reconstruída:\n", B)
```

Faz? um dot de Q com L e depois um dot com R

```
Matriz Original:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Matriz Original Reconstruída:
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

Exercício 8 - Você encontra abaixo o link de um repositório no Github para uma app em Python de uma Calculadora de Álgebra Linear, que apresenta via web a possibilidade de fazer muitos dos cálculos que estudamos até aqui no curso. Todo o código está em Python sendo uma excelente fonte de material de referência de estudo.

Faça o download do repositório para sua máquina e estude os códigos apresentados.

https://github.com/subram43/linear-algebra-calculator (https://github.com/subram43/linear-algebra-calculator)

Fim

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 9

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Cálculo Diferencial

Você pode usar métodos algébricos para calcular a taxa de mudança ao longo de um intervalo de função unindo dois pontos na função com uma linha e medindo sua inclinação. Por exemplo, uma função pode retornar a distância percorrida por um ciclista em um período de tempo, e você pode usar uma linha para medir a velocidade média entre dois pontos no tempo. No entanto, isso não diz a vecolidade do ciclista em um único ponto no tempo - apenas a velocidade média ao longo de um intervalo.

Para encontrar a velocidade do ciclista em um ponto específico no tempo, você precisa encontrar a inclinação de uma curva em um determinado ponto. *Cálculo Diferencial* nos permite fazer isso através do uso de *derivadas*. Podemos usar derivadas para encontrar a inclinação em um valor específico x calculando um delta para valores de x_1 e x_2 que são infinitesimamente próximos - então você pode pensar nisso como medir a inclinação de uma pequena linha reta que compreende parte da curva.

Introdução aos Limites

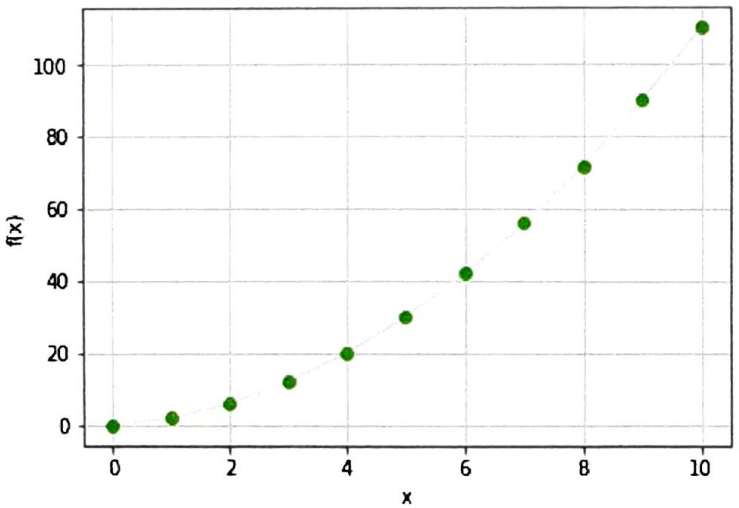
No entanto, antes de podermos pular direto para as derivadas (assunto do próximo capítulo), precisamos examinar outro aspecto do cálculo diferencial - o *limite* de uma função; o que nos ajuda a medir como o valor de uma função muda à medida que o valor x_2 se aproxima de x_1

Para entender melhor os limites, vamos observar mais de perto uma função e observar que, embora representemos a função como uma linha, ela é, na verdade, composta de pontos individuais. Execute a seguinte célula para mostrar os pontos que plotamos para valores inteiros de x - a linha é criada interpolando os pontos:

In [1]:

```
import matplotlib.pyplot as plt
%matplotlib inline

# Função
def f(x):
    return x**2 + x

# Cria uma lista de valores x de 0 a 10 para o plot
x = list(range(0, 11))

# Obtém os valores y correspondentes da função
y = [f(i) for i in x]

# Configura o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfacecolor='gree

plt.show()
```

Para cada Valor de i (Recebidos de x) em chama a função f(x)

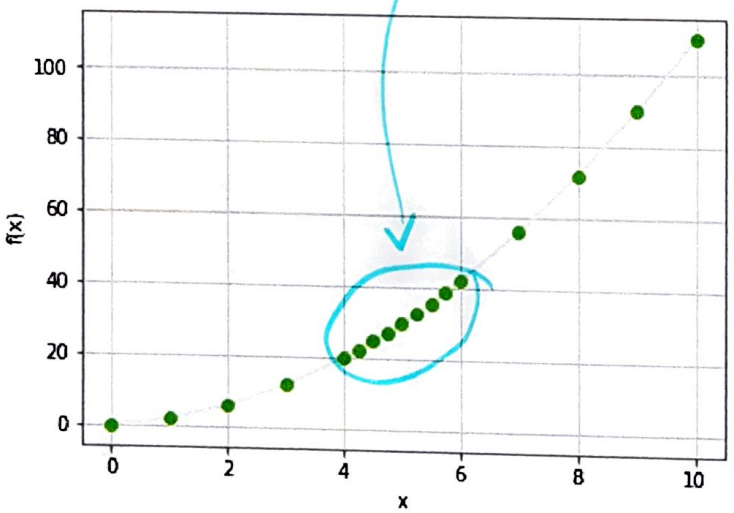


Sabemos a partir da função que os valores de ***f (x)*** são calculados calculando o quadrado do valor ***x*** e adicionando ***x*** . Para que possamos calcular facilmente os pontos e mostrá-los - execute o seguinte código:

In [2]:

```
import matplotlib.pyplot as plt

%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma lista de valores x de 0 a 10 para o plot
x = list(range(0,5))

x.append(4.25)
x.append(4.5)
x.append(4.75)
x.append(5)
x.append(5.25)
x.append(5.5)
x.append(5.75)
x = x + list(range(6,11))

# Obtém os valores y correspondentes da função

y = [f(i) for i in x]

# Configura o gráfico

plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfacecolor='gree

plt.show()
```

← Adicionando alguns pontos específicos



Agora podemos ver mais claramente que esta linha de função é formada por uma série contínua de pontos,

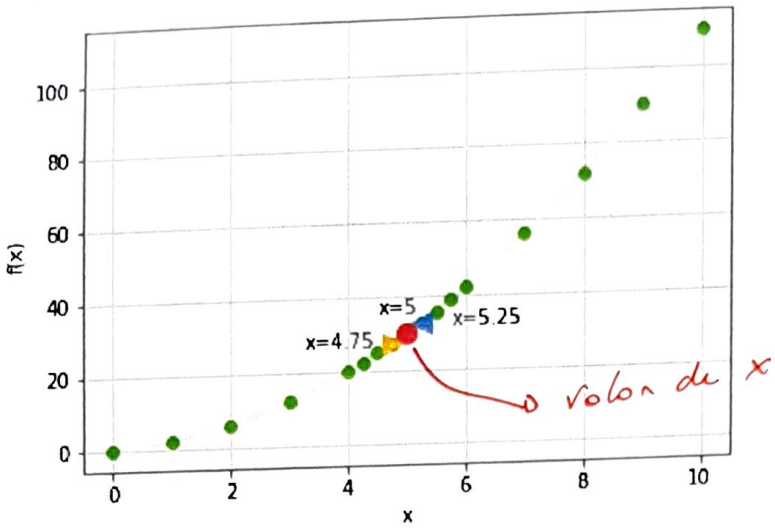
então, teoricamente, para qualquer valor dado de ***x*** há um ponto na linha, e há um ponto adjacente em ambos os lados com um valor que é o mais próximo possível de ***x***, mas na verdade não ***x***.

Execute o seguinte código para visualizar um ponto específico para $x = 5$ e tente identificar o ponto mais próximo de cada lado:

In [3]:

```
import matplotlib.pyplot as plt

%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma lista de valores x de 0 a 10 para o plot
x = list(range(0,5))

x.append(4.25)
x.append(4.5)
x.append(4.75)
x.append(5)
x.append(5.25)
x.append(5.5)
x.append(5.75)

x = x + list(range(6,11))

# Obtém os valores y correspondentes da função
y = [f(i) for i in x]

# Configura o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfacecolor='gree

zx = 5

zy = f(zx)

plt.plot(zx, zy, color='red', marker='o', markersize=10)

plt.annotate('x=' + str(zx),(zx, zy), xytext=(zx - 0.5, zy + 5))

# Plot f(x) quando x = 5.1
posx = 5.25

posy = f(posx)

plt.plot(posx, posy, color='blue', marker='<', markersize=10)

plt.annotate('x=' + str(posx),(posx, posy), xytext=(posx + 0.5, posy - 1))

# Plot f(x) quando x = 4.9
negx = 4.75

negy = f(negx)
```

```
plt.plot(negx, negy, color='orange', marker='>', markersize=10)

plt.annotate('x=' + str(negx),(negx, negy), xytext=(negx - 1.5, negy - 1))

plt.show()
```



Você pode ver o ponto onde ***x*** é 5, e você pode ver que há pontos mostrados no gráfico que parecem estar bem próximos a este ponto (em *x = 4.75* e *x = 5.25*). No entanto, se aumentássemos o zoom, veríamos que ainda existem lacunas que poderiam ser preenchidas por outros valores de ***x*** que estão ainda mais próximos de 5; por exemplo, 4.9 e 5.1, ou 4.999 e 5.001. Se pudéssemos fazer zoom infinitamente próximo à linha, veríamos que não importa o quão próximo um valor você usasse (por exemplo, 4.999999999999), sempre há um valor que é um pouco mais próximo (por exemplo, 4.9999999999999).

Então, o que podemos dizer é que existe um número hipotético que é o mais próximo possível do nosso valor desejado de *x* sem realmente ser *x* , mas não podemos expressá-lo como um número real. Em vez disso, expressamos simbolicamente como um *limite* , assim:

$$\boxed{\lim_{x \to 5} f(x)}$$

Isso é interpretado como *o limite da função f (x) quando x se aproxima de 5 .*

Limites e Continuidade

A função ***f (x)*** é *contínua* para todos os valores reais numerados de ***x*** . Simplificando, isso significa que você pode desenhar a linha criada pela função sem levantar sua caneta.

No entanto, isso não é necessariamente verdade de todas as funções. Considere a função ***g (x)*** abaixo:

$$\boxed{g(x) = -(\frac{12}{2x})^2}$$

Esta função é um pouco mais complexa do que a anterior, mas a principal coisa a observar é que ela requer uma divisão por *2x* . Agora, pergunte a si mesmo; o que aconteceria se você aplicasse essa função a um valor *x* de **0** ?

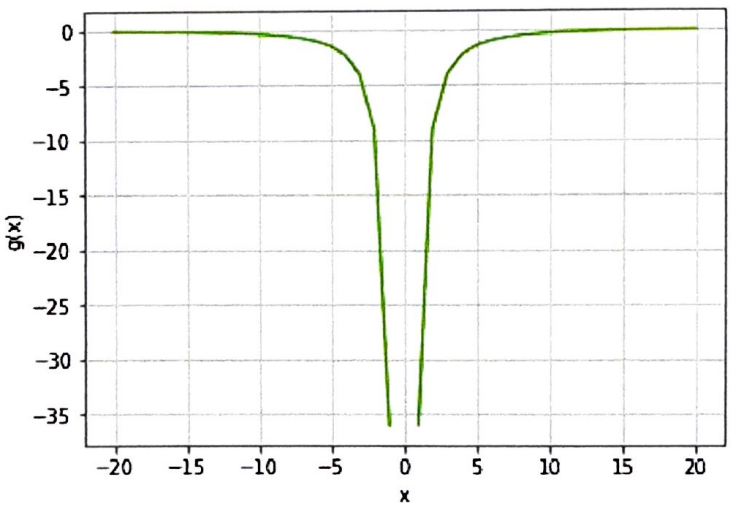
Qualquer coisa dividida por 0 é *indefinida* . Portanto, o *domínio* desta função não inclui 0; em outras palavras, a função é definida quando *x* é qualquer número real tal que *x não é igual a 0* . A função deve, portanto, ser escrita assim:

$$g(x) = -(\frac{12}{2x})^2, \quad x \neq 0$$

Então, por que isto é importante? Vamos investigar executando o seguinte código Python para definir a função e plotá-la para um conjunto arbitrário de valores:

In [4]:

```
import matplotlib.pyplot as plt
%matplotlib inline

# Função
def g(x):

    if x != 0:

        return -(12/(2*x))**2


# Cria uma lista de valores x
x = range(-20, 21)

# Obtém os valores y correspondentes da função
y = [g(a) for a in x]

# Configura o gráfico

plt.xlabel('x')

plt.ylabel('g(x)')

plt.grid()

# Plot x contra g (x)
plt.plot(x,y, color='green')

plt.show()
```



Não tem um contra domínio! não tem como ter um saído quando não tem x = 0

Olhe atentamente para o plot, e observe a lacuna da linha onde x = 0. Isso indica que a função não está definida aqui. O *domínio* da função (o conjunto de possíveis valores de entrada) não inclui 0, e o *range* (o conjunto de possíveis valores de saída) não inclui um valor para x = 0.

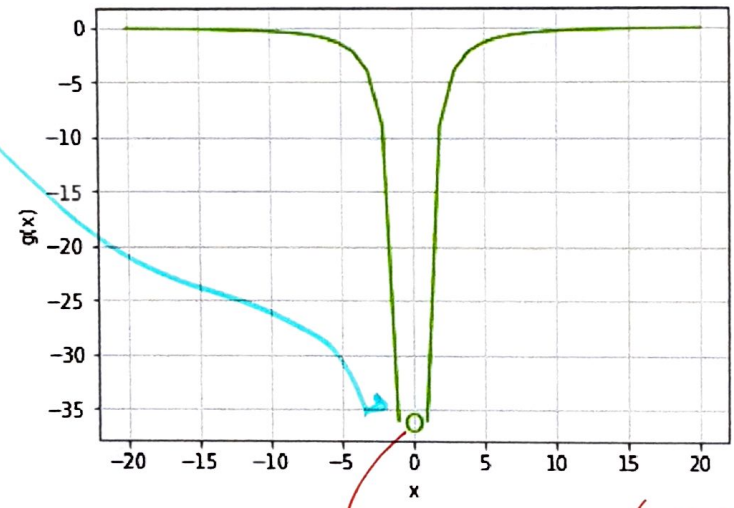
Esta é uma função *não-contínua* - em outras palavras, inclui pelo menos uma lacuna quando plotada (para que você não possa plotar manualmente sem levantar a caneta). Especificamente, a função é não contínua em x = 0.

Por convenção, quando uma função não contínua é plotada, os pontos que formam uma linha contínua (ou *intervalo*) são mostrados como uma linha, e o final de cada linha onde há uma descontinuidade é mostrado como um círculo, que é preenchido se o valor nesse ponto estiver incluído na linha e vazio se o valor não estiver incluído na linha.

Neste caso, a função produz dois intervalos com uma lacuna entre eles onde a função não está definida, então podemos mostrar o ponto descontínuo como um círculo não preenchido - execute o seguinte código para visualizar isso com o Python:

In [5]:

```
import matplotlib.pyplot as plt
%matplotlib inline


# Função
def g(x):

    if x != 0:

        return -(12/(2*x))**2

# Cria uma lista de valores x
x = range(-20, 21)

# Obtem os valores y correspondentes da função
y = [g(a) for a in x]

# Configura o gráfico
plt.xlabel('x')

plt.ylabel('g(x)')

plt.grid()

# Plot x contra g (x)
plt.plot(x,y, color='green')

# Plot de um círculo no gap
xy = (0,g(1))

plt.annotate('O',xy, xytext=(-0.7, -37),fontsize=14,color='green')

plt.show()
```



o círculo representa uma função não contínua.

Existem várias razões pelas quais uma função pode ser não contínua. Por exemplo, considere a seguinte função:

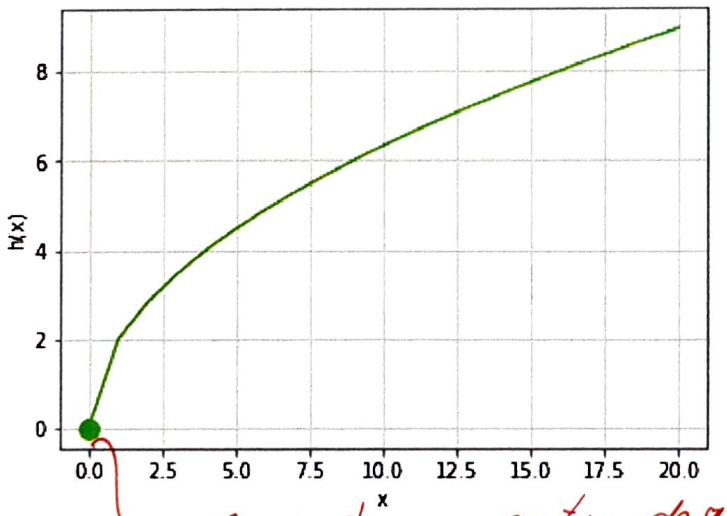
$$\boxed{h(x) = 2\sqrt{x}, \quad x \geq 0}$$

A aplicação desta função a um valor ***x*** não negativo retorna uma saída válida; mas para qualquer valor em que ***x*** é negativo, a saída é indefinida, porque a raiz quadrada de um valor negativo não é um número real.

Aqui está o código Python para plotar a função ***h*** :

In [6]:

```
import matplotlib.pyplot as plt
%matplotlib inline

# Função
def h(x):

    if x >= 0:

        import numpy as np

        return 2 * np.sqrt(x)

# Cria uma lista de valores x
x = range(-20, 21)

# Obtém os valores y correspondentes da função
y = [h(a) for a in x]

# Configura o gráfico
plt.xlabel('x')

plt.ylabel('h(x)')

plt.grid()

# Plot x contra h(x)

plt.plot(x,y, color='green')

# Plot de um círculo perto o suficiente do limite h (-x) para nossos propósitos!
plt.plot(0, h(0), color='green', marker='o', markerfacecolor='green', markersize=10)

plt.show()
```



Somente a partir de ~~zero~~ zero

Agora, suponha que tenhamos uma função assim:

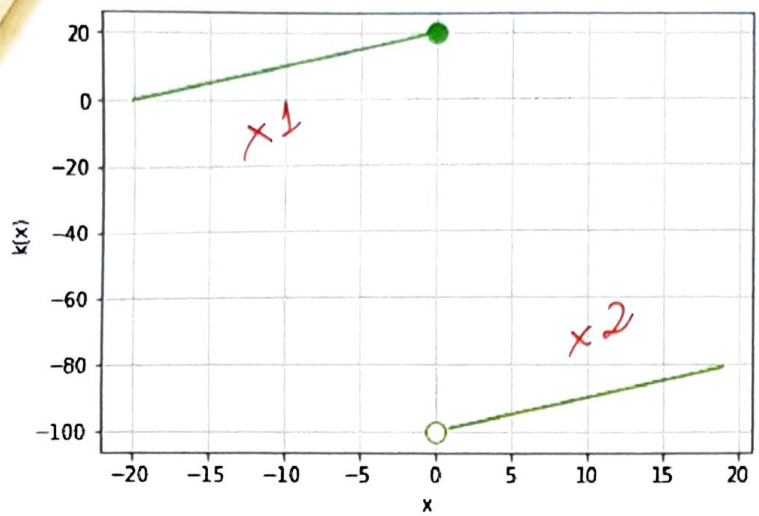
$$\boxed{k(x) = \begin{cases} x + 20, & \text{if } x \leq 0, \\ x - 100, & \text{Ou então} \end{cases}}$$

Nesse caso, o domínio da função inclui todos os números reais, mas sua saída ainda não é contínua devido à maneira como valores diferentes são retornados, dependendo do valor de x . O *intervalo* de saídas possíveis para *$k(x \leq 0)$ é ≤ 20, e o intervalo de valores de saída para $k\ (x > 0)$ é $x > -100$.

Vamos usar o Python para traçar a função ***k*** :

In [7]:

```
import matplotlib.pyplot as plt
%matplotlib inline

# Função
def k(x):

    import numpy as np

    if x <= 0:

        return x + 20

    else:

        return x - 100

# Cria uma lista de valores x para cada intervalo não contínuo
x1 = range(-20, 1)

x2 = range(1, 20)

# Obtém os valores y correspondentes da função
y1 = [k(i) for i in x1]

y2 = [k(i) for i in x2]

# Configura o gráfico
plt.xlabel('x')

plt.ylabel('k(x)')

plt.grid()

# Plot x contra k(x)
plt.plot(x1,y1, color='green')

plt.plot(x2,y2, color='green')

# Plot de um círculo onde termina o intervalo

plt.plot(0, k(0), color='green', marker='o', markerfacecolor='green', markersize=10)

plt.plot(0, k(0.0001), color='green', marker='o', markerfacecolor='w', markersize=10)

plt.show()
```



Exercício 1 - Escreva uma função em Python que represente a função abaixo. Passe 2 valores como parâmetro e teste a função.

$f(x) = x^3 - 5x^2 + 9$

In [8]:

```
# Solução

def f(x):
    return x**3 - 5*x**2 + 9

print(f(1), f(2))
```

Potência

```
5 -3
```

Exercício 2 - Escreva uma função em Python que represente a função exponencial abaixo. Teste sua função.

$exp(x) = e^x$ domínio igual a $(-\infty, \infty)$,range igual a $(0, \infty)$

In [9]:

```
# Solução

import numpy as np

def exp(x):
    return np.e**x

print("exp(2) = e^2 = ", exp(2))
```

Representa o Nº de Euler

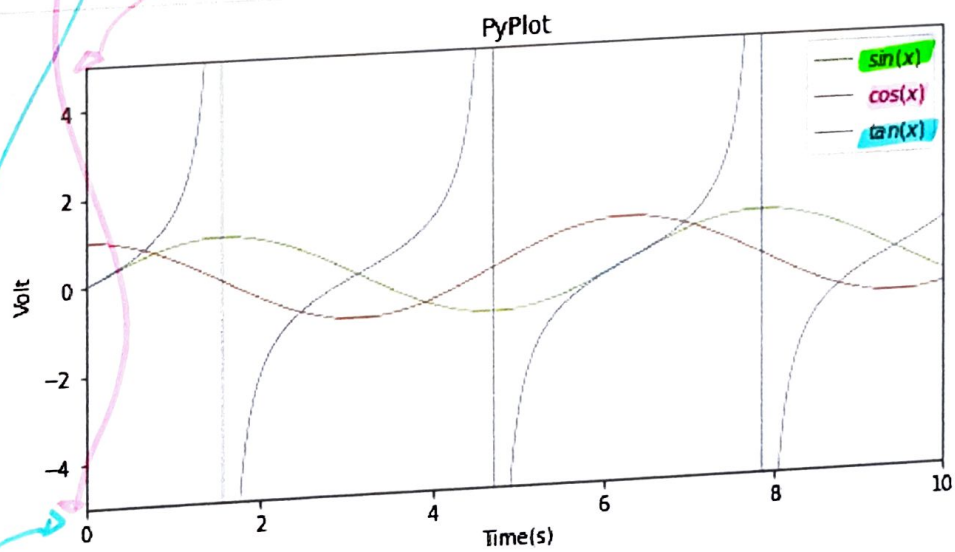
"e" é um método do numpy
Por isso np

```
exp(2) = e^2 =  7.3890560989306495
```

Exercício 3 - Coloque um único plot o resultado das 3 funções trigonométricas estudadas neste capítulo.

In [10]:

```
# Solução

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(-10, 10, 10000)

a = np.sin(x)

b = np.cos(x)

c = np.tan(x)

plt.figure(figsize=(8,4))
plt.plot(x,a,label='$sin(x)$',color='green',linewidth=0.5)

plt.plot(x,b,label='$cos(x)$',color='red',linewidth=0.5)

plt.plot(x,c,label='$tan(x)$',color='blue',linewidth=0.5)

plt.xlabel('Time(s)')

plt.ylabel('Volt')

plt.title('PyPlot')

plt.xlim(0,10)

plt.ylim(-5,5)

plt.legend()

plt.show()
```



Exercício 4 - Considere as funções abaixo. Represente-as em três funções Python distintas.

$(x) = x^2, g(x) = x^2 + x, h(x) = x^4 + 2x^2 \cdot x + x^2$

In [11]:

```
# Solução

def f(x):
    return x**2

def g(x):
    return x**2+x

def h(x):
    return f(g(x))

print("f(1) equals",f(1),"g(1) equals",g(1),"h(1) equals",h(1))

x = np.array(range(-10,10))

y = np.array([h(i) for i in x])

plt.scatter(x,y,)
```

$x = 1 = 1^4 + 2 \cdot 1^2 \cdot 1 + 1^2 =$

$1 + 2 \cdot 1 + 1 =$

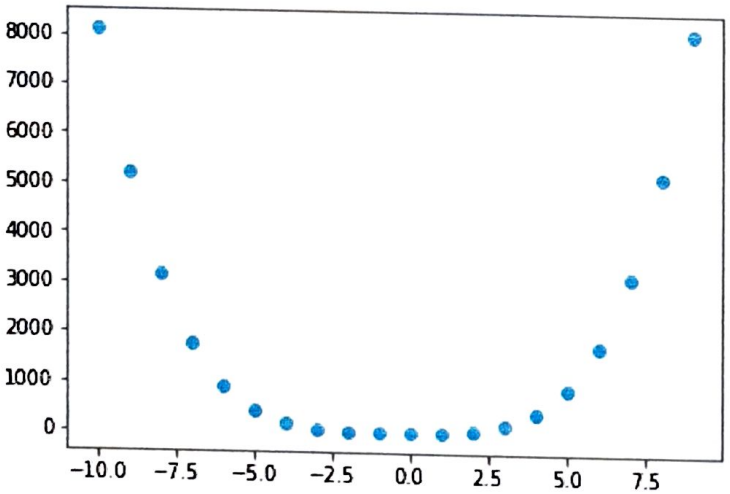
$1 + 2 + 1 = 4$

→ para cada x
Chamar a função de H

```
f(1) equals 1 g(1) equals 2 h(1) equals 4
```

Out[11]:

```
<matplotlib.collections.PathCollection at 0x11483d240>
```



Exercício 5 - Crie um plot para mostrar o valor de retorno da função f quando x for igual a 0.

$$f(x) = x^2 + 1$$

In [12]:

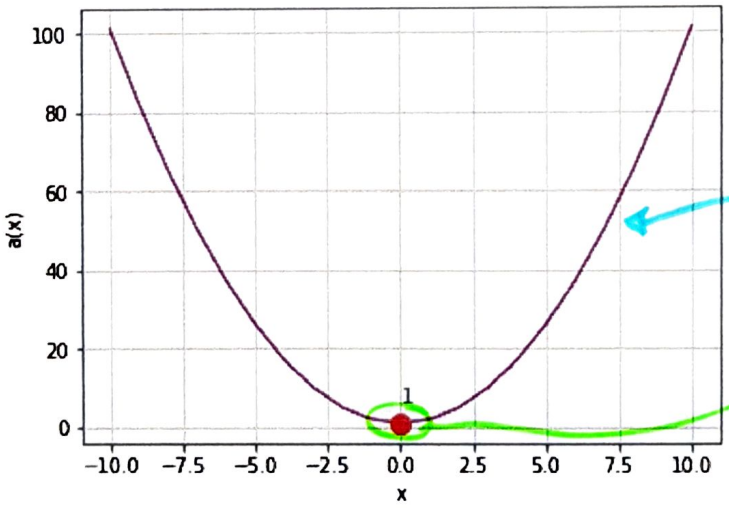
```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def f(x):
    return x**2 + 1

# Cria uma lista de valores x
x = range(-10, 11)

# Obtém os valores y correspondentes da função
y = [f(i) for i in x]

# Cria o gráfico
plt.xlabel('x')

plt.ylabel('a(x)')

plt.grid()

# Plot x contra f(x)
plt.plot(x,y, color='purple')

# Plot f(x) quando x = 0
zx = 0
zy = f(zx)

plt.plot(zx, zy, color='red', marker='o', markersize=10)

plt.annotate(str(zy),(zx, zy), xytext=(zx, zy + 5))

plt.show()
```

Cria a função

list comprehension
chama a função p/ cada valor de x

Quando x = 0 chama a função

Mas todos os valores são plotados



Exercício 6 - O que acontece se usarmos valores x levemente maiores ou menores que 0? Crie um gráfico e descubra!

$$\boxed{f(x) = x^2 + 1}$$

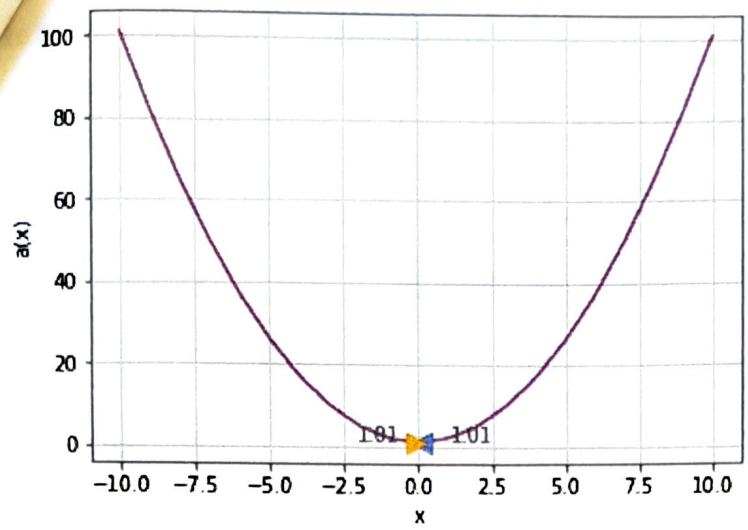
Esses valores ***x*** retornam valores de ***f (x)*** que estão um pouco acima de 1, e se fôssemos continuar plotando números que estão cada vez mais próximos de 0, por exemplo, 0,0000000001 ou -0,0000000001, a função ainda retornaria um valor que é apenas ligeiramente maior que 1. O limite de função *f (x)* como *x* se aproxima de 0, é 1; e a notação para indicar isso é:

$$\boxed{\lim_{x \to 0} f(x) = 1}$$

Isso reflete uma definição mais formal de continuidade de função. Anteriormente, afirmamos que uma função é contínua em um ponto se você puder desenhá-la nesse ponto sem levantar sua caneta. A definição mais matemática é que uma função é contínua em um ponto se o limite da função à medida que se aproxima desse ponto de ambas as direções é igual ao valor da função naquele ponto. Neste caso, quando nos aproximamos de x = 0 de ambos os lados, o limite é 1; e o valor de *f (0)* também é 1; então a função é contínua em x = 0.

In [13]:

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def a(x):
    return x**2 + 1

# Cria uma lista de valores x
x = range(-10, 11)

# Obtém os valores y correspondentes da função
y = [a(i) for i in x]

# Cria o gráfico
plt.xlabel('x')
plt.ylabel('a(x)')
plt.grid()

# Plot x contra a(x)
plt.plot(x,y, color='purple')

# Plot a(x) quando x = 0.1
posx = 0.1

posy = a(posx)

plt.plot(posx, posy, color='blue', marker='<', markersize=10)

plt.annotate(str(posy),(posx, posy), xytext=(posx + 1, posy))

# Plot a(x) quando x = -0.1
negx = -0.1

negy = a(negx)

plt.plot(negx, negy, color='orange', marker='>', markersize=10)

plt.annotate(str(negy),(negx, negy), xytext=(negx - 2, negy))

plt.show()
```



Exercício 7 - Implemente a função abaixo em Python e crie um plot para demonstrar o resultado da execução do seu programa.

$$\boxed{b(x) = -2x^2 \cdot \frac{1}{x}, \quad x \neq 0}$$

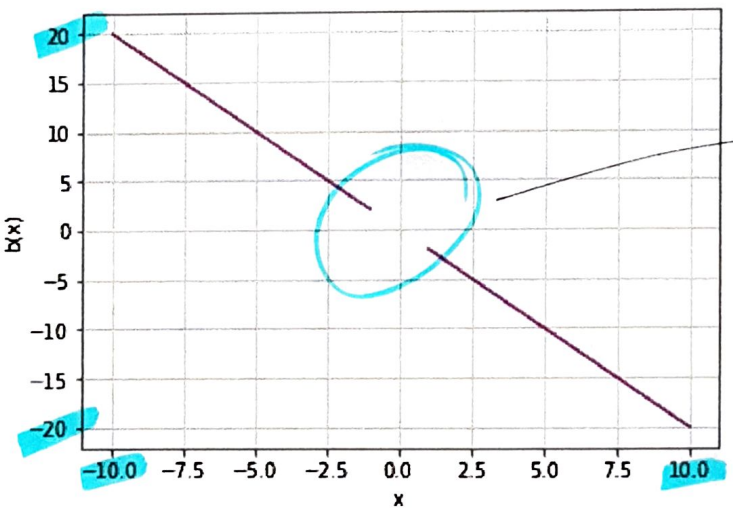
A saída desta função contém uma lacuna na linha onde x = 0. Parece que não apenas o *domínio* da função (os valores que podem ser passados como *x*) excluem 0; mas o *range* da função (o conjunto de valores que podem ser retornados a partir dele) também exclui 0.

In [14]:

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x

# Cria uma lista de valores
x = range(-10, 11)

# Obtém os valores y correspondentes da função
y = [b(i) for i in x]

# Cria o gráfico
plt.xlabel('x')

plt.ylabel('b(x)')

plt.grid()

# Plot x contra b(x)
plt.plot(x,y, color='purple')

plt.show()
```

x deve ser diferente de ∅

= -20 → 20



Não tem zero

Exercício 8 - No Exercício 7 não podemos avaliar a função para um valor x de 0, mas podemos ver o que ela retorna para um valor que é apenas um pouco menor que 0? Se a resposta for positiva, mostre isso em código Python.

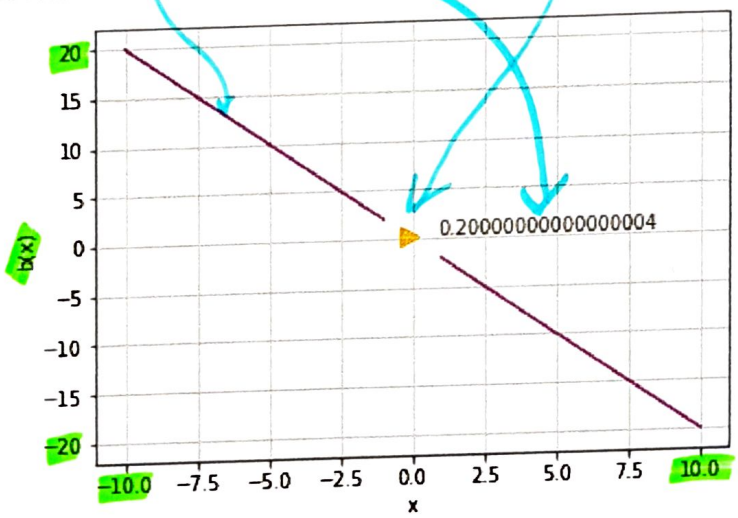
In [15]:

Limite de Função

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x

# Cria uma lista de valores
x = range(-10, 11)

# Obtém os valores y correspondentes da função
y = [b(i) for i in x]

# Cria o gráfico
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x contra b(x)
plt.plot(x,y, color='purple')

# Plot b(x) para x = -0.1
negx = -0.1

negy = b(negx)

plt.plot(negx, negy, color='orange', marker='>', markersize=10)

plt.annotate(str(negy),(negx, negy), xytext=(negx + 1, negy))

plt.show()
```

Se não tem o domínio zero não temos como gerar um contra domínio

Podemos representar um valor muito próximo de zero, que não é zero



Exercício 9 - Podemos tentar um valor negativo *x* que seja um pouco mais próximo de 0 considerando a função anterior? Como você explicaria isso? Justifique sua resposta e crie o código em Python.

Como o valor de *x* se aproxima de 0 da esquerda (negativo), o valor de *b (x)* está diminuindo em direção a 0. Podemos mostrar isso com a seguinte notação:

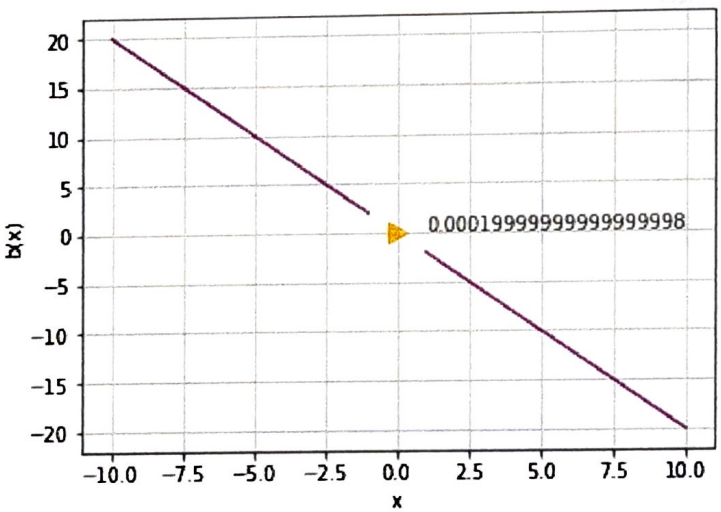
$$\boxed{\lim_{x \to 0^-} b(x) = 0}$$

o limite de x tende a ∅ considerando x próximo de ∅

Observe que a seta aponta para 0 ⁻ (com um sinal de menos) para indicar que estamos descrevendo o limite à medida que nos aproximamos de 0 do lado negativo.

In [16]:

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x

# Cria uma lista de valores
x = range(-10, 11)

# Obtém os valores y correspondentes da função
y = [b(i) for i in x]

# Cria o gráfico
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x contra b(x)
plt.plot(x,y, color='purple')

# Plot b(x) para x = -0.0001
negx = -0.0001

negy = b(negx)

plt.plot(negx, negy, color='orange', marker='>', markersize=10)

plt.annotate(str(negy),(negx, negy), xytext=(negx + 1, negy))

plt.show()
```



Encontrando Limites Utilizando uma Tabela

Até agora, estimamos os limites de um ponto graficamente examinando um gráfico de uma função. Você também pode aproximar os limites criando uma tabela de valores x e os valores de função correspondentes de cada lado do ponto para o qual você deseja encontrar os limites.

Por exemplo, vamos voltar para a nossa função ***f*** :

$$\boxed{f(x) = x^2 + 1}$$

Se quisermos encontrar os limites como x está se aproximando de 0, podemos aplicar a função a alguns valores de cada lado de 0 e visualizá-los como uma tabela. Consegue fazer isso em Python?

Exercício 10 - Crie uma tabela com os limites da função acima.

In [17]:

```
# Solução

import pandas as pd

# Função
def f(x):
    return x**2 + 1

# Crie um dataframe com uma coluna x contendo valores de cada lado de 0
df = pd.DataFrame ({'x': [-1, -0.5, -0.2, -0.1, -0.01, 0, 0.01, 0.1, 0.2, 0.5, 1]})

# Adicione uma coluna f(x) aplicando a função a x
df['f(x)'] = f(df['x'])

# Imprime o resultado
df
```

Similar ao excel

Chave

Valores

dicionário

Pego cada elemento em 'x' para passar p/ função.

Out[17]:

	x	f(x)
0	-1.00	2.0000
1	-0.50	1.2500
2	-0.20	1.0400
3	-0.10	1.0100
4	-0.01	1.0001
5	0.00	1.0000
6	0.01	1.0001
7	0.10	1.0100
8	0.20	1.0400
9	0.50	1.2500
10	1.00	2.0000

Cria uma outra coluna 'f(x)' no df

df['x']

Chave

Nome da coluna

Olhando para a saída, você pode ver que os valores da função estão se aproximando de 1 quando x se aproxima de 0 de ambos os lados, então:

$$\lim_{x \to 0} f(x) = 1$$

Além disso, você pode ver que o valor real da função quando x = 0 também é 1, portanto:

$$\boxed{\lim_{x \to 0} f(x) = f(0)}$$

Que de acordo com nossa definição anterior, significa que a função é contínua em 0.

No entanto, você deve ter cuidado para não assumir que o limite quando x está se aproximando de 0 será sempre o mesmo que o valor quando x = 0; mesmo quando a função é definida para x = 0.

Exercício 11 (Desafio) - Considere a seguinte função abaixo. Crie uma tabela com os limites da função.

$$\boxed{e(x) = \begin{cases} 5, & \text{if } x = 0, \\ 1 + x^2, & \text{ou então} \end{cases}}$$

Se x = 0 retorno 5
ou então
$1 + x^2$

In [18]:

```
# Solução

import pandas as pd

# Função
def e(x):
    if x == 0:
        return 5
    else:
        return 1 + x**2

# Crie um dataframe com uma coluna x contendo valores de cada lado de 0
x= [-1, -0.5, -0.2, -0.1, -0.01, 0, 0.01, 0.1, 0.2, 0.5, 1]

# Gera os valores de saída
y =[e(i) for i in x]

# Cria o dataframe
df = pd.DataFrame ({' x':x, 'e(x)': y })

# Imprime o resultado
df
```

Se x = ϕ retorna 5

Se não Retorno 1 + x²

chave valor chave valor

Out[18]:

	x	e(x)
0	-1.00	2.0000
1	-0.50	1.2500
2	-0.20	1.0400
3	-0.10	1.0100
4	-0.01	1.0001
5	0.00	5.0000
6	0.01	1.0001
7	0.10	1.0100
8	0.20	1.0400
9	0.50	1.2500
10	1.00	2.0000

Como antes, você pode ver que, conforme os valores *x* se aproximam de 0 de ambos os lados, o valor da função se aproxima de 1, então:

$$\lim_{x \to 0} e(x) = 1$$

No entanto, o valor real da função quando x = 0 é 5, não 1; assim:

$$\lim_{x \to 0} e(x) \neq e(0)$$

Que de acordo com nossa definição anterior, significa que a função é não contínua em 0.

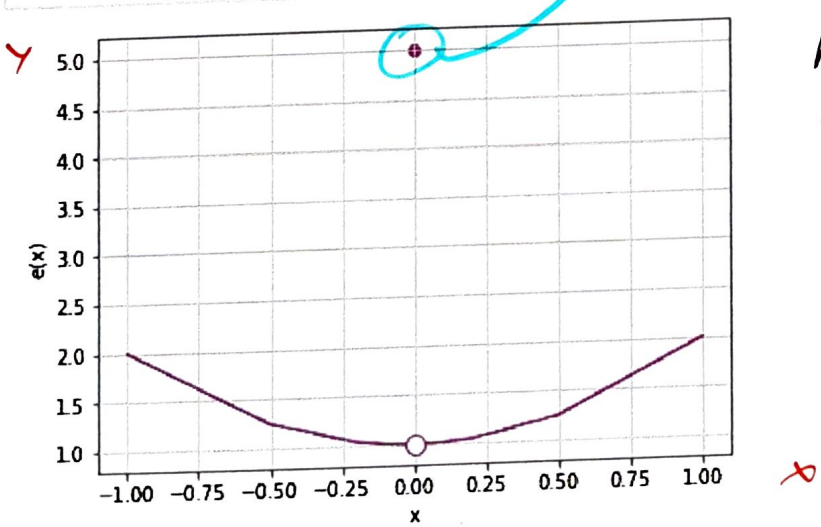
Para encerrar, consegue representar isso em um gráfico?

Exercício 12 (Desafio Master) - Crie um gráfico que represente a tabela do exercício anterior.

In [19]:

```
# Solução

import matplotlib.pyplot as plt
%matplotlib inline

# Função
def e(x):
    if x == 0:
        return 5
    else:
        return 1 + x**2

    # Crie um dataframe com uma coluna x contendo valores de cada lado de 0
x= [-1, -0.5, -0.2, -0.1, -0.01, 0.01, 0.1, 0.2, 0.5, 1]

# Gera os valores de saída
y =[e(i) for i in x]

# Cria o gráfico
plt.xlabel('x')
plt.ylabel('e(x)')
plt.grid()

# Plot x contra e(x)

plt.plot(x, y, color='purple')

plt.scatter(0, e(0), color='purple')

plt.plot(0, 1, color='purple', marker='o', markerfacecolor='w', markersize=10)

plt.show()
```

Valores de x e y



Função não contínua
os pontos não se encontram

Data Science Academy

Matemática Para Machine Learning

Lista de Exercícios - Capítulo 10

O objetivo desta lista de exercícios é você praticar os principais conceitos estudados neste capítulo, ao mesmo tempo que desenvolve suas habilidades em lógica de programação com a linguagem Python.

Caso tenha dúvidas, isso é absolutamente normal e faça um trabalho de pesquisa a fim de relembrar o formato das operações matemáticas.

Quando encontrar o formato de uma operação que resolva o exercício proposto, use a linguagem Python para representar esta operação. Em essência, é assim que aplicamos Matemática Para Machine Learning, construindo algoritmos e representando esses algoritmos em linguagem de programação.

Divirta-se!!

Cálculo Diferencial

Até agora, você aprendeu como avaliar limites para pontos em uma linha. Agora você vai desenvolver esse conhecimento e olhar para uma técnica de cálculo chamada *diferenciação* . Na diferenciação, usamos nosso conhecimento de limites para calcular a *derivada* de uma função para determinar a taxa de mudança em um ponto individual em uma linha.

Vamos nos lembrar do problema que estamos tentando resolver, aqui está uma função:

$$\boxed{f(x) = x^2 + x}$$

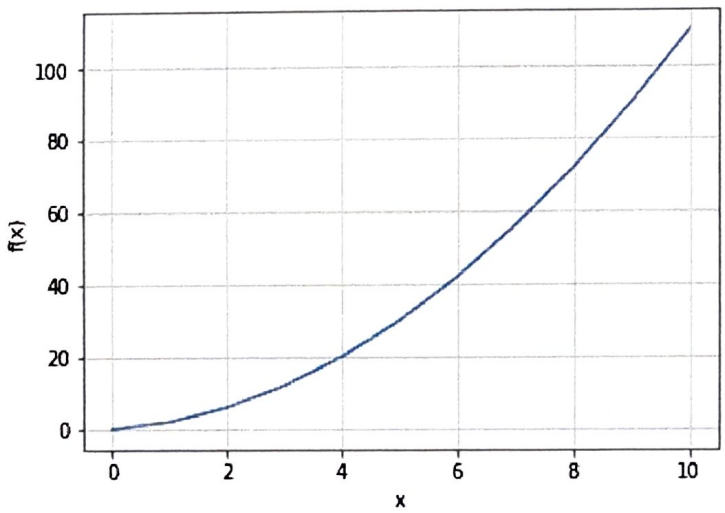
Podemos visualizar parte da linha que esta função define usando o seguinte código Python:

In [1]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y

y = [f(i) for i in x]

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot
plt.plot(x,y, color='blue')

plt.show()
```

Para no 10

Para cada valor de x eu calculo o y



Agora, sabemos que podemos calcular a taxa média de mudança para um determinado intervalo na linha calculando a inclinação de uma linha secante que conecta dois pontos na linha. Por exemplo, podemos calcular a mudança média para o intervalo entre x = 4 e x = 6 dividindo a mudança (ou *delta* , indicado como Δ) no valor de *f (x)* pela mudança no valor de *x* :

$$m = \frac{\Delta f(x)}{\Delta x}$$

O delta para *f (x)* é calculado subtraindo os valores *f (x)* de nossos pontos, e o delta para *x* é calculado subtraindo os valores *x* de nossos pontos, assim:

$$m = \frac{f(x)_2 - f(x)_1}{x_2 - x_1}$$

Então, para o intervalo entre x = 4 e x = 6, temos:

$$m = \frac{f(6) - f(4)}{6 - 4}$$

Podemos calcular e plotar isso usando o seguinte código Python:

In [2]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y
y = [f(i) for i in x]

# Define os valores
x1 = 4

x2 = 6

# Obter os valores f (x) correspondentes
y1 = f(x1)

y2 = f(x2)

# Calcula a inclinação dividindo os deltas
a = (y2 - y1)/(x2 - x1)

# Cria uma matriz de valores x para a linha secante
sx = [x1,x2]

# Use a função para obter os valores y
sy = [f(i) for i in sx]

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da Função

plt.plot(x,y, color='blue')

# Plot dos intervalos

plt.scatter([x1,x2],[y1,y2], c='red')

# Plot da linha secante

plt.plot(sx,sy, color='magenta')

# Exibir a taxa média de mudança que foi calculada

plt.annotate('Mudança Média =' + str(a),(x2, (y2+y1)/2))
```

$4^2 = 16 + 4 = 20$

$6^2 = 36 + 6 = 42$

$42 - 20 = 22$

$6 - 4 = 2$ segundos

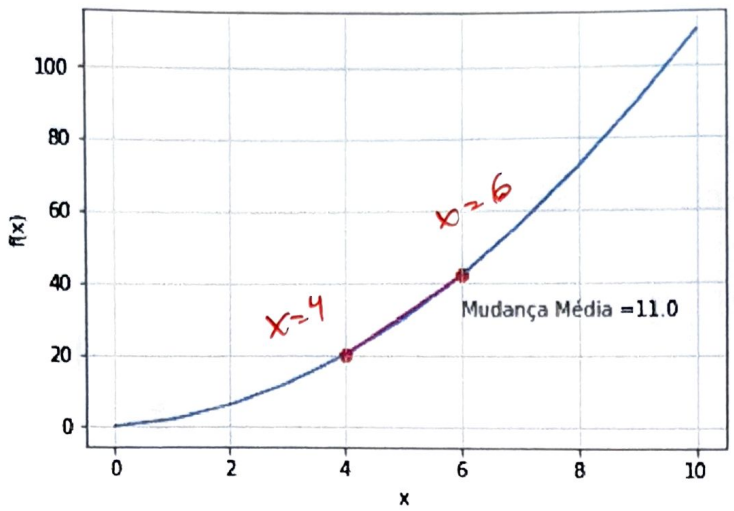
22 metro / 2 seg

= 11 metros por seg

$m = \frac{\Delta f(x)}{\Delta (x)}$

?

```
plt.show()
```



A taxa média de mudança para o intervalo entre x = 4 e x = 6 é $^{11}/_{1}$ (ou simplesmente 11), significando que para cada **1** adicionado para *x* , *f(x)* aumenta em **11** .

Em outras palavras, se x representa o tempo em segundos e f(x) representa a distância em metros, a taxa média de mudança da distância ao longo do tempo (em outras palavras, *velocidade*) para o intervalo de 4 a 6 segundos é de 11 metros por segundo.

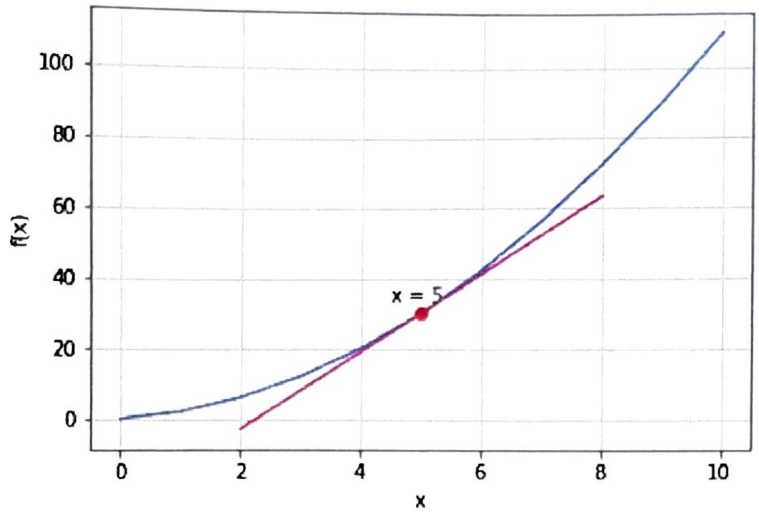
Até agora, isso é apenas álgebra básica; mas e se em vez da taxa média de mudança ao longo de um intervalo, queremos calcular a taxa de mudança em um único ponto, digamos, onde x = 5?

Uma abordagem que podemos usar é criar uma linha secante entre o ponto em que queremos o declive e outro ponto na linha de função que está infinitamente próximo a ela. Tão perto, na verdade, que a linha secante é, na verdade, uma tangente que passa por ambos os pontos. Podemos então calcular a inclinação para a linha secante como antes. Isso seria parecido com o gráfico produzido pelo seguinte código:

In [3]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y

y = [f(i) for i in x]

# Defina o ponto x1, arbitrariamente 5

x1 = 5

y1 = f(x1)

# Defina o ponto x2, muito próximo de x1

x2 = 5.000000001

y2 = f(x2)

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot
plt.plot(x,y, color='blue')

# Plot do ponto

plt.scatter(x1,y1, c='red')

plt.annotate('x = ' + str(x1),(x1,y1), xytext=(x1-0.5, y1+3))

# Aproxima o declive da tangente e plote-o
m = (y2-y1)/(x2-x1)

xMin = x1 - 3

yMin = y1 - (3*m)

xMax = x1 + 3

yMax = y1 + (3*m)

plt.plot([xMin,xMax],[yMin,yMax], color='magenta')
```

```
plt.show()
```



é bom para entender o conceito, mas na prática é pouco usual

Calculando a Derivada

No código Python acima, criamos a linha secante (quase) tangencial especificando um ponto que está muito próximo do ponto em que queremos calcular a taxa de mudança. Isso é adequado para mostrar a linha conceitualmente no gráfico, mas não é uma maneira particularmente generalizável (ou precisa) de realmente calcular a linha para que possamos obter a taxa de alteração em qualquer ponto.

Se pelo menos soubéssemos uma maneira para calcular um ponto na linha o mais próximo possível de apontar com um determinado valor x .

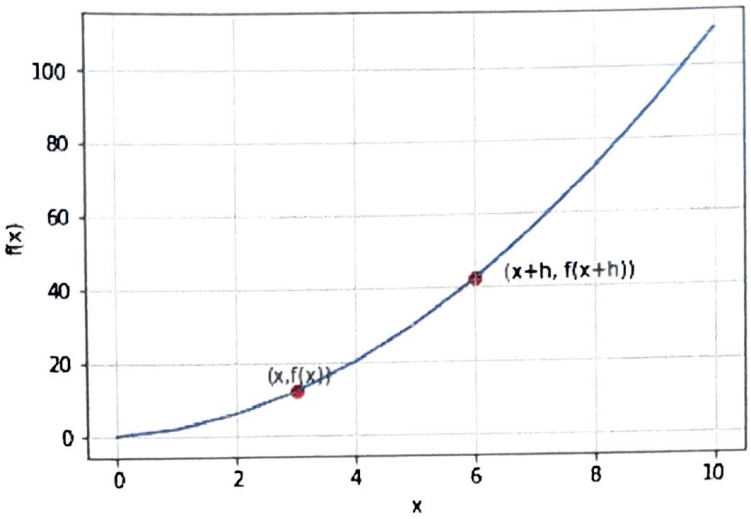
Não, espere, nós já sabemos! É um limite :-).

Então, como podemos aplicar um limite neste cenário? Bem, vamos começar examinando nossa abordagem geral para calcular a inclinação um pouco mais detalhadamente. Nossa abordagem testada e comprovada é plotar uma linha secante entre dois pontos com valores diferentes de x, então vamos plotar uma linha arbitrária (x, y) e, em seguida, adicionar um valor arbitrário para x , que chamaremos de h . Então sabemos que podemos traçar uma linha secante entre ($x, f(x)$) e ($x + h, f(x + h)$) e encontrar sua inclinação.

Execute a célula abaixo para ver esses pontos:

In [4]:

```
from matplotlib import pyplot as plt

%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y

y = [f(i) for i in x]

# Defina o ponto x1 e calcula y1

x1 = 3

y1 = f(x1)

# Define o incremento

h = 3

# Configura o ponto x +h

x2 = x1+h

y2 = f(x2)

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot
plt.plot(x,y, color='blue')

# Plot do ponto x, y
plt.scatter(x1,y1, c='red')

plt.annotate('(x,f(x))',(x1,y1), xytext=(x1-0.5, y1+3))

# Plot do ponto x + h
plt.scatter(x2,y2, c='red')

plt.annotate('(x+h, f(x+h))',(x2,y2), xytext=(x2+0.5, y2))

plt.show()
```



Como vimos anteriormente, nossa fórmula para calcular a inclinação é:

$$m = \frac{\Delta f(x)}{\Delta x}$$

O delta para *f (x)* é calculado subtraindo os valores *f (x + h)* e *f (x)* dos nossos pontos, e o delta para *x* é apenas a diferença entre *x* e *x + h* ; em outras palavras, *h* :

$$m = \frac{f(x+h) - f(x)}{h}$$

O que realmente precisamos é da inclinação na menor distância possível entre x e x + h, então estamos procurando o menor valor possível de *h* . Em outras palavras, precisamos do limite quando *h* se aproxima 0.

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Esta equação é generalizável, e podemos usá-la como a definição de uma função para nos ajudar a encontrar a inclinação em qualquer valor dado de *x* na linha, e é o que chamamos de *derivada* de nossa função original (que neste caso é chamado *f'*). Isto é geralmente indicado na notação *Lagrange* desta forma:

função derivada de x →

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Às vezes, você também verá derivadas escritas na notação *Leibniz* desta forma:

função derivada de x →

$$\frac{d}{dx} f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Nota: * Alguns livros usam **h** para simbolizar a diferença entre x_0 e x_1, enquanto outros usam **Δ x** . Não faz diferença o valor simbólico que você usa.

Forma Alternativa Para Uma Derivada

A fórmula acima mostra a forma generalizada de uma derivada. Você pode usar a função derivada para obter o declive em qualquer ponto dado, por exemplo, para obter o declive no ponto *a* você poderia apenas plugar o valor para *a* na função derivada generalizada:

$$f'(\mathbf{a}) = \lim_{h \to 0} \frac{f(\mathbf{a}+h) - f(\mathbf{a})}{h}$$

Ou você poderia usar a forma alternativa, que é específica para apontar *a* :

$$\boxed{f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}}$$

Estes são matematicamente equivalentes.

Encontrando a Derivada Para Um Ponto Específico

É mais fácil entender a diferenciação vendo-a em ação, então vamos usá-la para encontrar a derivada de um ponto específico na função ***f*** .

Aqui está a definição de função ***f*** :

$$\boxed{f(x) = x^2 + x}$$

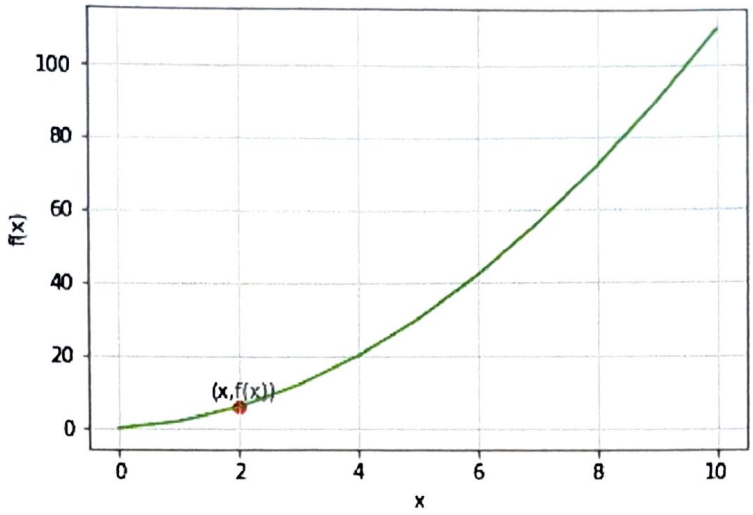
Vamos dizer que queremos encontrar ***f '(2)*** (a derivada para ***f*** quando ***x*** é 2); então estamos tentando encontrar a inclinação no ponto mostrado pelo seguinte código:

Exercício 1 - Onde estiver com a palavra RESOLVER, você deve preencher com o código correspondente.

In [5]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y
# RESOLVER

y = [f(i) for i in x]

# Define os pontos
# RESOLVER

x1 = 2

y1 = f(x1)

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

# Plot do ponto

plt.scatter(x1,y1, c='red')

plt.annotate('(x,f(x))',(x1,y1), xytext=(x1-0.5, y1+3))

plt.show()
```

$x = 2$



Aqui está nossa fórmula generalizada para encontrar uma derivada em um ponto específico (*a*):

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

Então vamos começar ligando nosso valor *a* em:

$$f'(2) = \lim_{h \to 0} \frac{f(2+h) - f(2)}{h}$$

Sabemos que ***f (x)*** encapsula a equação $x^2 + x$, então podemos reescrever nossa equação derivada como:

Onde for $f(x)$
$= x^2 + x$

$$f'(2) = \lim_{h \to 0} \frac{((2+h)^2 + 2 + h) - (2^2 + 2)}{h}$$

Sendo $x = 2$

Podemos aplicar a propriedade de distribuição para $(2 + h)^2$ usando a regra que $(a + b)^2 = a^2 + b^2 + 2ab$:

$$f'(2) = \lim_{h \to 0} \frac{(4 + h^2 + 4h + 2 + h) - (2^2 + 2)}{h}$$

Então podemos simplificar $2^2 + 2$ (2^2 é 4, mais 2 dá 6):

$$f'(2) = \lim_{h \to 0} \frac{(4 + h^2 + 4h + 2 + h) - 6}{h}$$

Podemos combinar termos semelhantes no lado esquerdo do numerador para tornar as coisas um pouco mais claras:

$$f'(2) = \lim_{h \to 0} \frac{(h^2 + 5h \cancel{+ 6}) \cancel{- 6}}{h}$$

Que combina ainda mais para se livrar do *6* :

$$f'(2) = \lim_{h \to 0} \frac{h^2 + 5\cancel{h}}{\cancel{h}}$$

E finalmente, podemos simplificar a fração:

$$f'(2) = \lim_{h \to 0} h + 5$$

Para obter o limite quando *h* está se aproximando de 0, podemos usar a substituição direta para h:

$$f'(2) = 0 + 5$$

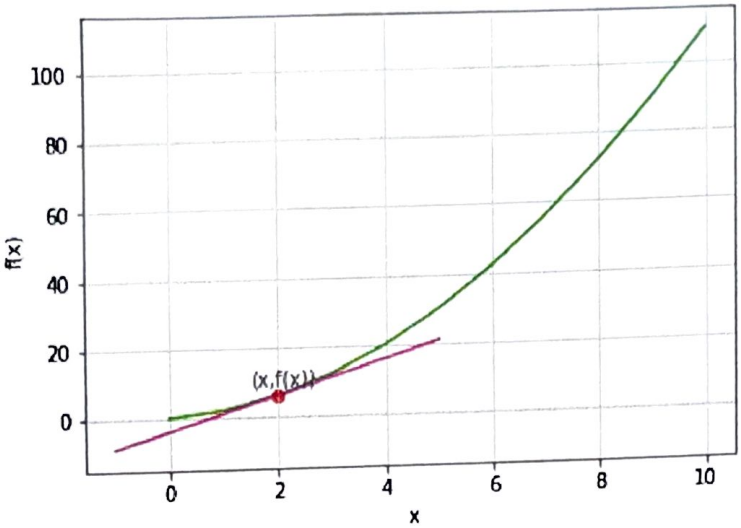
assim:

$$f'(2) = 5$$ → valor da derivada

Vamos desenhar uma linha tangente com a inclinação em nosso gráfico para ver se está certo:

In [6]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y

y = [f(i) for i in x]

# Define os pontos
x1 = 2

y1 = f(x1)

# Especifique a derivada que calculamos acima
m = 5

# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

# Plot do ponto
plt.scatter(x1,y1, c='red')

plt.annotate('(x,f(x))',(x1,y1), xytext=(x1-0.5, y1+3))

# Plot da linha tangente usando a derivada que calculamos
xMin = x1 - 3

yMin = y1 - (3*m)

xMax = x1 + 3

yMax = y1 + (3*m)

plt.plot([xMin,xMax],[yMin,yMax], color='magenta')

plt.show()
```



Conseguimos encontrar a derivada e a linha tangente que a representa

Como Encontrar a Derivada Para Qualquer Ponto

Agora vamos juntar tudo e definir uma função que podemos usar para encontrar a derivada para qualquer ponto na função ***f*** :

Aqui está nossa função derivada geral novamente:

$$\boxed{f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

Sabemos que ***f (x)*** encapsula a equação ***x*** ^{2} ***+ x*** , então podemos reescrever nossa equação derivada como:

$$f'(x) = \lim_{h \to 0} \frac{((x+h)^2 + x + h) - (x^2 + x)}{h}$$

Podemos aplicar a propriedade de distribuição para ***(x + h)*** ^{2} usando a regra que *(a + b)* ^{2} *= a* ^{2} *+ b* ^{2} *+ 2ab* :

$$f'(x) = \lim_{h \to 0} \frac{(x^2 + h^2 + 2xh + x + h) - (x^2 + x)}{h}$$

Então podemos usar a propriedade distributiva para expandir ***- (x*** ^{2} ***+ x)*** , que é a mesma coisa que *-1 (x* ^{2} *+ x)* , para - ***x*** ^{2} - ***x*** :

$$f'(x) = \lim_{h \to 0} \frac{\cancel{x^2} + h^2 + 2xh + \cancel{x} + h - \cancel{x^2} - \cancel{x}}{h}$$

Podemos combinar termos semelhantes no numerador para tornar as coisas um pouco mais claras:

$$f'(x) = \lim_{h \to 0} \frac{h^2 + 2xh + h}{h}$$

? Porque h^2 virou 1?

$0^2 = 1$

E finalmente, podemos simplificar a fração:

$$f'(x) = \lim_{h \to 0} 2x + h + 1$$

Para obter o limite quando *h* está se aproximando de 0, podemos usar a substituição direta para h:

$$f'(x) = 2x + 0 + 1$$

assim:

$$\boxed{f'(x) = 2x + 1}$$

Forma simplificada
executamos essa função p/ encontrar a derivada

Agora temos uma função para a derivada de ***f***, que podemos aplicar a qualquer valor *x* para encontrar a inclinação da função em ***f (x)***.

Por exemplo, vamos encontrar a derivada de ***f*** com um valor *x* de 5:

$$f'(5) = 2 \cdot 5 + 1 = 10 + 1 = 11$$

Vamos usar Python para definir as funções ***f (x)*** e ***f '(x)***, plotar ***f (5)*** e mostrar a linha tangente para ***f '(5)***.

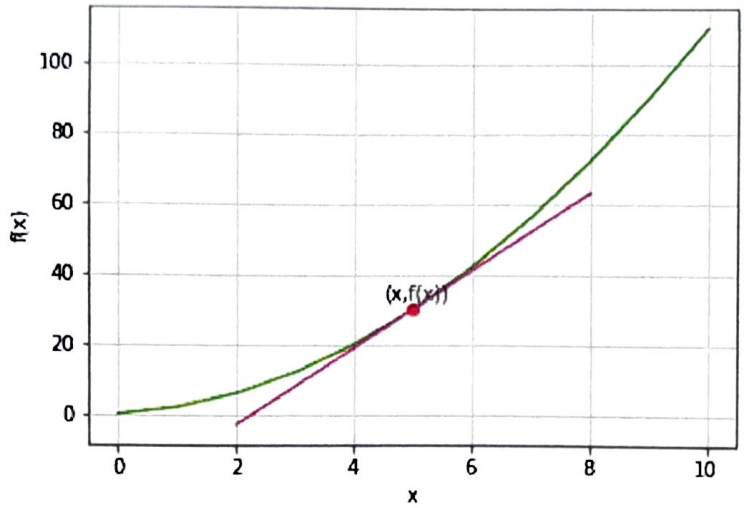
Exercício 2 - Onde estiver com a palavra RESOLVER, você deve preencher com o código correspondente.

In [7]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):

    return x**2 + x

# Função derivada para f
# RESOLVER
def fd(x):

    return (2 * x) + 1

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(0, 11))

# Use a função para obter os valores y

y = [f(i) for i in x]

# Define o ponto

x1 = 5

y1 = f(x1)

# Calcula a derivada usando a função derivada

m = fd(x1)

# Define o gráfico

plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot da função
plt.plot(x,y, color='green')

# Plot do ponto
plt.scatter(x1,y1, c='red')

plt.annotate('(x,f(x))',(x1,y1), xytext=(x1-0.5, y1+3))

# Plot da linha tangente usando a derivada que calculamos
xMin = x1 - 3

yMin = y1 - (3*m)

xMax = x1 + 3

yMax = y1 + (3*m)
```

Sempre que tivermos a função x^2+x eu só preciso aplicar a função $2 \cdot x + 1$ que consigo encontrar a derivada de qualquer ponto

```
plt.plot([xMin,xMax],[yMin,yMax], color='magenta')

plt.show()
```



Derivadas de Equações

Nós temos falado sobre derivadas de *funções* , mas é importante lembrar que funções são apenas operações nomeadas que retornam um valor. Podemos aplicar o que sabemos sobre o cálculo de derivadas para qualquer equação, por exemplo:

$$\frac{d}{dx}(2x+6)$$

Note que geralmente mudamos para a notação *Leibniz* ao encontrar derivadas de equações que não são encapsuladas como funções; mas a abordagem para resolver este exemplo é exatamente a mesma que se tivéssemos uma função hipotética com a definição *2x + 6* :

H tendendo a ϕ

$$\frac{d}{dx}(2x+6) = \lim_{h\to 0}\frac{(2(x+h)+6)-(2x+6)}{h}$$

$$\frac{F(x_2 - x_1)}{x_2 - x_1}$$

Após fatorar o *2 (x + h)* à esquerda e o *- (2x - 6)* à direita, isto é:

$$\frac{d}{dx}(2x+6) = \lim_{h\to 0}\frac{2x+2h+6-2x-6}{h}$$

Podemos simplificar isso para:

$$\frac{d}{dx}(2x+6) = \lim_{h\to 0}\frac{2h}{h}$$

Agora podemos fatorar *h* completamente, então em qualquer ponto:

$$\frac{d}{dx}(2x+6) = 2$$

Se você executar o código Python abaixo para traçar a linha criada pela equação, verá que ela realmente tem uma inclinação constante de 2.

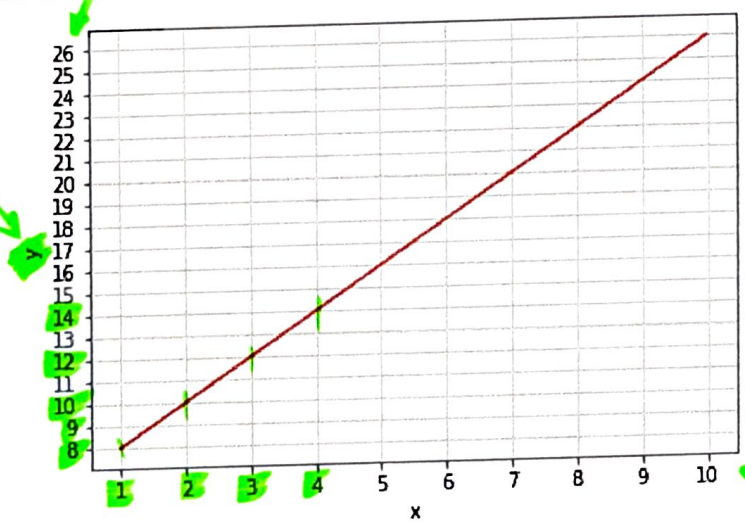
Exercício 3 - Onde estiver com a palavra RESOLVER, você deve preencher com o código correspondente.

In [8]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Cria uma matriz de valores x de 0 a 10 para o plot.

x = list(range(1, 11))

# Use a função para obter os valores y
# RESOLVER

y = [(2*i) + 6 for i in x]

# Define o gráfico

plt.xlabel('x')

plt.xticks(range(1,11, 1))

plt.ylabel('y')

plt.yticks(range(8,27, 1))

plt.grid()

# Plot
plt.plot(x,y, color='red')


plt.show()
```

$\frac{d}{dx}(2x+6)=2$

i representa cada elemento na lista de valores x



$2\times1+6=8$

$2\times2+6=10$

$2\times3+6=12$

$2\times4+6=14$

Regras e Operações com Derivadas

Ao trabalhar com derivadas, existem algumas regras ou atalhos que você pode aplicar para facilitar sua vida.

Regras Básicas de Derivadas

Vamos começar com algumas regras básicas que é útil saber.

- Se $f(x) = C$ (onde C é uma constante), então $f'(x) = 0$

  Isso faz sentido se você pensar nisso por um segundo. Não importa o valor que você usa para x, a função retorna o mesmo valor constante; então o gráfico da função será uma linha horizontal. Não há taxa de variação em uma linha horiziontal, então sua inclinação é 0 em todos os pontos. Isto é verdade para qualquer constante, incluindo constantes simbólicas como π (pi).

  Então, por exemplo:

  constante = linha Horizontal
  variação = ϕ

$$f(x) = 6 \quad \therefore \quad f'(x) = 0$$

  Ou:

$$f(x) = \pi \quad \therefore \quad f'(x) = 0$$

- Se $f(x) = Cg(x)$, então $f'(x) = Cg'(x)$

  Esta regra nos diz que se uma função é igual a uma segunda função multiplicada por uma constante, então a derivada da primeira função será igual à derivada da segunda função multiplicada pela mesma constante. Por exemplo:

$$\boxed{f(x) = 2g(x) \quad \therefore \quad f'(x) = 2g'(x)}$$

- Se $f(x) = g(x) + h(x)$, então $f'(x) = g'(x) + h'(x)$

  Em outras palavras, se uma função é a soma de duas outras funções, então a derivada da primeira função é a soma das derivadas das outras duas funções. Por exemplo:

$$f(x) = g(x) + h(x) \quad \therefore \quad f'(x) = g'(x) + h'(x)$$

  Soma do funções é igual a soma do derivada dos funções

  Claro, isso também se aplica à subtração:

$$f(x) = k(x) - l(x) \quad \therefore \quad f'(x) = k'(x) - l'(x)$$

Como discutido anteriormente, as funções são apenas equações encapsuladas como uma entidade nomeada que retorna um valor e as regras podem ser aplicadas a qualquer equação. Por exemplo:

$$\frac{d}{dx}(2x + 6) = \frac{d}{dx}2x + \frac{d}{dx}6$$

Assim, podemos aproveitar as regras para facilitar o cálculo:

$$\frac{d}{dx}(2x) = \lim_{h \to 0} \frac{2(x + h) - 2x}{h}$$

Após fatorar o $2(x + h)$ à esquerda, isto é:

$$\frac{d}{dx}(2x) = \lim_{h \to 0} \frac{2x + 2h - 2x}{h}$$

Podemos simplificar isso para:

$$\frac{d}{dx}(2x) = \lim_{h \to 0} \frac{2h}{h}$$

O que nos dá:

$$\frac{d}{dx}(2x) = 2$$

Agora podemos voltar nossa atenção para a derivada da constante 6 em relação a x , e sabemos que a derivada de uma constante é sempre 0, então:

$$\frac{d}{dx}(6) = 0$$

Adicionamos as duas derivadas que calculamos:

$$\frac{d}{dx}(2x + 6) = 2 + 0$$

O que nos dá o nosso resultado:

$$\boxed{\frac{d}{dx}(2x + 6) = 2}$$

A Regra da Potência

A *regra da potência* é um dos atalhos mais úteis no mundo do cálculo diferencial. Pode ser dito assim:

$$\boxed{f(x) = x^n \quad \therefore \quad f'(x) = nx^{n-1}}$$

Então, se a nossa função para x retorna x para a potência de alguma constante (que vamos chamar de n), então a derivada da função para x é n vezes x para o poder de n - 1.

Provavelmente é útil examinar alguns exemplos para ver como isso funciona:

$$f(x) = x^3 \quad \therefore \quad f'(x) = 3x^2$$

$$f(x) = x^{-2} \quad \therefore \quad f'(x) = -2x^{-3}$$

$$f(x) = x^2 \quad \therefore \quad f'(x) = 2x$$

Em cada um desses exemplos, o exponencial de x na definição da função torna-se o coeficiente para x na definição de derivada, com o exponencial sendo decrementado por 1.

Aqui está um exemplo trabalhado para encontrar a derivada da seguinte função:

$$f(x) = x^2$$

Então começamos com a função derivada geral:

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

Nós podemos conectar nossa definição para f :

$$f'(x) = \lim_{h \to 0} \frac{(x+h)^2 - x^2}{h}$$

Agora podemos fatorar o binômio quadrado perfeito à esquerda:

$$f'(x) = \lim_{h \to 0} \frac{\cancel{x^2} + h^2 + 2xh - \cancel{x^2}}{h}$$

Os termos x^2 cancelam um ao outro, então chegamos a:

$$f'(x) = \lim_{h \to 0} \frac{h^2 + 2xh}{h}$$

?

Por que h^2 vira h?

O que simplifica para:

$$f'(x) = \lim_{h \to 0} h + 2x$$

Com *h* se aproximando de 0, isto é:

$$f'(x) = 0 + 2x$$

Então nossa resposta é:

$$f'(x) = 2x$$

Note que poderíamos ter alcançado o mesmo resultado simplesmente aplicando a regra da potência e transformando x^2 em $2x^1$ (que é o mesmo que 2x).

A Regra do Produto

A *regra do produto* pode ser declarada como:

$$\boxed{\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + f(x)g'(x)}$$

Isso significa que a derivada de *f (x)* multiplicada por *g (x)* é igual à derivada de *f (x)* multiplicada pelo valor de *g (x)* adicionado ao valor de *g (x) f (x)* multiplicado pela derivada de *g (x)*.

Vamos ver um exemplo baseado nas duas funções a seguir:

$$f(x) = 2x^2$$

$$g(x) = x + 1$$

$2(x+h) = 2x^2 + 2h^2$

$2(x+h)^2 = 4xh$

Vamos começar calculando a derivada de *f (x)* :

$$f'(x) = \lim_{h \to 0} \frac{2(x+h)^2 - 2x^2}{h}$$

Isso fatora para:

$$f'(x) = \lim_{h \to 0} \frac{\cancel{2x^2} + 2h^2 + 4xh - \cancel{2x^2}}{h}$$

Que quando cancelamos $2x^2$ e $-2x^2$ é:

$$f'(x) = \lim_{h \to 0} \frac{2h^2 + 4xh}{h}$$

$$\frac{h^2}{h} = \frac{h \cdot \cancel{h}}{\cancel{h}} = h$$

O que simplifica para:

$$f'(x) = \lim_{h \to 0} 2h + 4x$$

Com *h* se aproximando de 0, isto é:

$$\boxed{f'(x) = 4x}$$

Agora vamos olhar para *g '(x)* :

$$g'(x) = \lim_{h \to 0} \frac{(x+h) + 1 - (x+1)}{h}$$

$$-(+x) = -x$$
$$-(+1) = -1$$

Podemos apenas remover os colchetes à esquerda e fatorar o - *(x + 1)* à direita:

$$g'(x) = \lim_{h \to 0} \frac{\cancel{x} + h + \cancel{1} - \cancel{x} - \cancel{1}}{h}$$

Qual pode ser simplificado até:

$$g'(x) = \lim_{h \to 0} \frac{h}{h}$$

Habilitando-nos a fatorar *h* completamente para dar uma derivada constante de *1* :

$$\boxed{g'(x) = 1}$$

Então, agora podemos calcular a derivada para o produto dessas funções, ligando as funções e as derivadas que calculamos para elas na equação da regra do produto:

$$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + f(x)g'(x)$$

Assim:

$$\frac{d}{dx}[f(x)g(x)] = (\boxed{4x} \cdot (x + 1)) + (2x^2 \cdot 1)$$

Que pode ser simplificado para:

$$\frac{d}{dx}[f(x)g(x)] = (4x^2 + 4x) + 2x^2$$

Que pode ser ainda mais simplificado para:

$$\frac{d}{dx}[f(x)g(x)] = 6x^2 + 4x$$

A Regra do Quociente

A *regra do quociente* aplica-se a funções que são definidas como um quociente de uma expressão dividida por outra, por exemplo:;

$$r(x) = \frac{s(x)}{t(x)}$$

Nesta situação, você pode aplicar a seguinte regra de quociente para encontrar a derivada de *r (x)* :

$$r'(x) = \frac{s'(x)t(x) - s(x)t'(x)}{[t(x)]^2}$$

Aqui estão as nossas definições para *s (x)* e *t (x)* :

$$s(x) = 3x^2$$

$$t(x) = 2x$$

Vamos começar com *s'(x)* :

$3(x+h)^2 = 3x^2 + 3h^2$

$3 \cdot 3 = 6xh$

$$s'(x) = \lim_{h \to 0} \frac{3(x+h)^2 - 3x^2}{h}$$

Isso fatora para:

$$s'(x) = \lim_{h \to 0} \frac{\cancel{3x^2} + 3h^2 + 6xh - \cancel{3x^2}}{h}$$

Que quando cancelamos $3x^2$ e $-3x^2$ é:

$$s'(x) = \lim_{h \to 0} \frac{3h^2 + 6xh}{h}$$

?

O que simplifica para:

$$s'(x) = \lim_{h \to 0} 3h + 6x$$

$h = \phi$

Com *h* se aproximando de 0, isto é:

$$s'(x) = 6x$$

Agora vamos olhar para *t'(x)* :

$$t'(x) = \lim_{h \to 0} \frac{2(x+h) - 2x}{h}$$

Podemos apenas fatorar o *2 (x + h)* à esquerda:

$$t'(x) = \lim_{h \to 0} \frac{2x + 2h - 2x}{h}$$

Qual pode ser simplificado até:

$$t'(x) = \lim_{h \to 0} \frac{2h}{h}$$

Habilitando-nos a fatorar *h* completamente para dar uma derivada constante de *2* :

$$t'(x) = 2$$

Então, agora podemos calcular a derivada para o quociente dessas funções, ligando as definições de função e as derivadas que calculamos para elas na equação da regra do quociente:

$$r'(x) = \frac{(6x \cdot 2x) - (3x^2 \cdot 2)}{[2x]^2}$$

Podemos fatorar os termos do numerador assim:

$$r'(x) = \frac{12x^2 - 6x^2}{[2x]^2}$$

$[2x]^2 = 2x \cdot 2x$

$[a \cdot b]^2 = (a \cdot b) \cdot (a \cdot b)$

Que então pode ser combinado:

$$r'(x) = \frac{6x^2}{[2x]^2}$$

O denominador é $[2x]^2$ (note que isso é diferente de $2x^2$. $[2x]^2$ é 2x & bull; 2x, enquanto $2x^2$ é $2 \cdot x^2$):

$$r'(x) = \frac{6x^2}{4x^2}$$

$\longrightarrow 6/4 = 1/\frac{1}{2}$

O que simplifica para:

$$r'(x) = 1\frac{1}{2}$$

Então a derivada de *r (x)* é 1.5.

A Regra da Cadeia

A *regra da cadeia* aproveita o fato de que as equações podem ser encapsuladas como funções, e como as funções contêm equações, é possível aninhar uma função dentro da outra.

Por exemplo, considere a seguinte função:

$$u(x) = 2x^2$$

Poderíamos ver a definição de *u(x)* como um composto de duas funções; uma função *inner* que calcula x^2 e uma função *outer* que multiplica o resultado da função interna por 2.

$$u(x) = 2(x^2)$$

Para tornar as coisas mais simples, podemos nomear essas funções internas e externas:

$$i(x) = x^2$$

$$o(x) = 2x$$

Note que x indica a entrada para cada função. A função *i* recebe sua entrada e eleva ao quadrado, e a função *o* pega sua entrada e a multiplica por 2. Quando usamos isso como uma função composta, a entrada do valor *x* na função externa será a saída da função interna.

Vamos dar uma olhada em como podemos aplicar essas funções para voltar à nossa função *u* original:

$$\boxed{u(x) = o(i(x))}$$

Então, primeiro precisamos encontrar a saída da função interna *i* para que possamos usar como o valor de entrada para a função externa *o* . Bem, isso é fácil, nós sabemos a definição de *i* (o quadrado da entrada), então podemos simplesmente ligá-lo:

$$u(x) = o(x^2)$$

Também conhecemos a definição para a função externa *o* (multiplique a entrada por 2), para que possamos aplicar isso à entrada:

$$u(x) = 2x^2$$

OK, agora sabemos como formar uma função composta. A *regra da cadeia* pode ser declarada assim:

$$\boxed{\frac{d}{dx}[o(i(x))] = o'(i(x)) \cdot i'(x)}$$

Tudo bem, vamos começar ligando a saída da função interna *i (x)* em:

$$\frac{d}{dx}[o(i(x))] = o'(x^2) \cdot i'(x)$$

Agora vamos usar isso para calcular a derivada de *o* , substituindo cada *x* na equação com a saída da função *i* (x^2):

?

$$o'(x) = \lim_{h \to 0} \frac{2(x^2 + h) - 2x^2}{h}$$

Isso fatora para:

$$o'(x) = \lim_{h \to 0} \frac{\cancel{2x^2} + 2h - \cancel{2x^2}}{h}$$

Que quando cancelamos $2x^2$ e $-2x^2$ é:

$$o'(x) = \lim_{h \to 0} \frac{2\cancel{h}}{\cancel{h}}$$

O que simplifica para:

$$\boxed{o'(x) = 2}$$

Agora podemos calcular *i '(x)* . Sabemos que a definição de *i (x)* é x^2 , e podemos usar a regra da potência para determinar que *i '(x)* é, portanto, 2x.

Então nossa equação neste ponto é:

$$\frac{d}{dx}[o(i(x))] = 2 \cdot 2x$$

A qual é:

$$\boxed{\frac{d}{dx}[o(i(x))] = 4x}$$

Comumente, a regra da cadeia é declarada usando uma notação ligeiramente diferente com a qual você pode achar mais fácil trabalhar. Nesse caso, podemos pegar nossa equação:

$$\frac{d}{dx}[o(i(x))] = o'(i(x)) \cdot i'(x)$$

e reescrevê-lo como

$$\frac{du}{dx} = \frac{do}{di}\frac{di}{dx}$$

Podemos então completar os cálculos da seguinte forma:

$$\frac{du}{dx} = 2 \cdot 2x = 4x$$

importante

Pontos Críticos e Otimização

Nós exploramos várias técnicas que podemos usar para calcular a derivada de uma função em um valor específico x ; em outras palavras, podemos determinar o declive da linha criada pela função em qualquer ponto da linha.

Essa capacidade de calcular a inclinação significa que podemos usar derivadas para determinar algumas propriedades interessantes da função.

Direção da Função em um Ponto

Considere a seguinte função, que representa a trajetória de uma bola que foi chutada em um campo de futebol:

$$k(x) = -10x^2 + 100x + 3$$

Execute o código Python abaixo para representar graficamente essa função e ver a trajetória da bola ao longo de um período de 10 segundos.

o que é otimização
o que representa
e como aplicar otimização.

In [9]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Cria a Função
def k(x):

    return -10*(x**2) + (100*x)  + 3

# Cria uma matriz de valores x para o plot

x = list(range(0, 11))


# Use a função para obter os valores y

y = [k(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em segundos)')

plt.ylabel('k(x) (altura em centímetros)')

plt.xticks(range(0,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot
plt.plot(x,y, color='green')

plt.show()
```

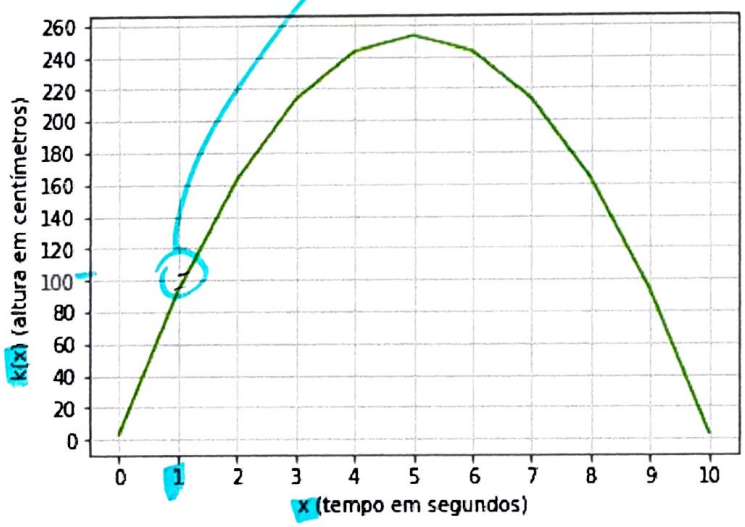
$\rightarrow -10 \cdot x^2 + 100 \cdot x + 3$

$x = 1$

$-10 \cdot 1^2 + 100 \cdot 1 + 3$

$-10 \cdot 1 + 100 \cdot 1 + 3$

$-10 + 100 + 3 = \boxed{97}$



Ao olhar para o gráfico desta função, você pode ver que ela descreve uma parábola na qual a bola subiu em altura antes de cair de volta ao chão. No gráfico, é bastante fácil ver quando a bola estava subindo e quando estava caindo.

Naturalmente, também podemos usar derivadas para determinar a inclinação da função em qualquer ponto. Podemos aplicar algumas das regras que discutimos anteriormente para determinar a função derivada:

- Podemos somar as derivadas dos termos individuais (- ***10x*** ^{2} , ***100x*** e ***3***) para encontrar a derivada da função inteira.
- A *regra da potência* nos diz que a derivada de - ***10x*** ^{2} é - ***10 • 2x*** , que é - ***20x*** .
- A *regra da potência* também nos diz que a derivada de ***100x*** é ***100*** .
- A derivada de qualquer constante, como ***3*** é ***0*** .

Assim:

$$k'(x) = -20x + 100 + 0$$

O que obviamente simplifica para:

$$\boxed{k'(x) = -20x + 100}$$

Agora podemos usar essa função derivada para encontrar a inclinação para qualquer valor de ***x*** .

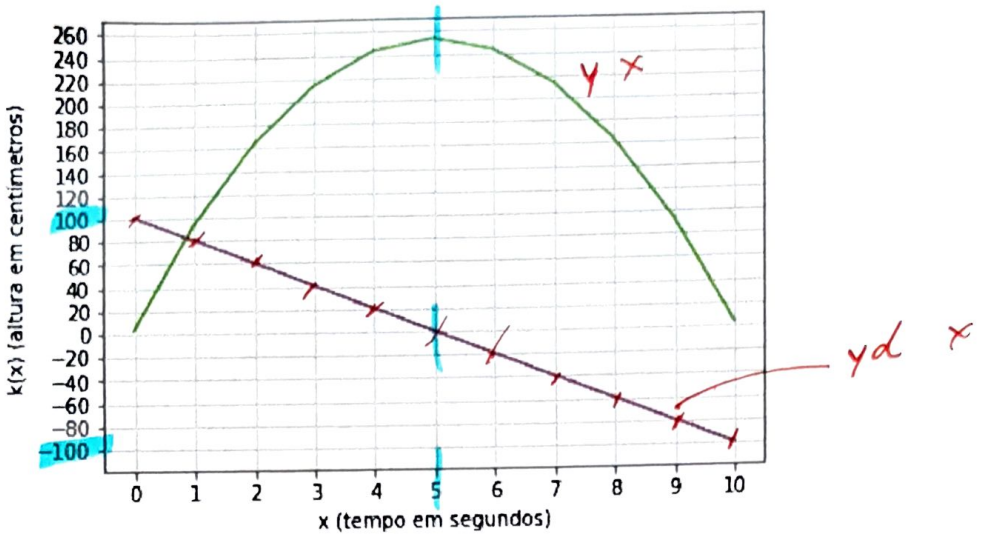
Execute a célula abaixo para ver um gráfico da função e sua função derivada:

In [10]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def k(x):

    return -10*(x**2) + (100*x)  + 3

def kd(x):

    return -20*x + 100

# Cria uma matriz de valores x para o plot

x = list(range(0, 11))

# Use a função para obter os valores y

y = [k(i) for i in x]

# Use a função derivada para obter as derivadas
yd = [kd(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em segundos)')

plt.ylabel('k(x) (altura em centímetros)')

plt.xticks(range(0,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

# Plot da derivada

plt.plot(x,yd, color='purple')

plt.show()
```

→ função derivada

$-20 \cdot 1 + 100 = 80$
$-20 \cdot 2 + 100 = 60$
$-20 \cdot 3 + 100 = 40$
$-20 \cdot 4 + 100 = 20$
$-20 \cdot 5 + 100 = 0$
$-20 \cdot 6 + 100 = -20$



Observe atentamente a linha roxa que representa a função derivada e observe que ela é um valor decrescente constante - em outras palavras, a inclinação da função está reduzindo linearmente à medida que x aumenta. Mesmo que o próprio valor da função esteja aumentando para a primeira metade da parábola (enquanto a bola está subindo), a inclinação está se tornando menos íngreme (a bola não está subindo a uma taxa tão alta), até que finalmente a bola atinge seu apogeu e a inclinação torna-se negativa (a bola começa a cair).

Observe também que o ponto em que a linha derivada cruza 0 no eixo y é também o ponto em que o valor da função pára de aumentar e começa a diminuir. Quando a inclinação tem um valor positivo, a função está aumentando; e quando a inclinação tem um valor negativo, a função está diminuindo.

O fato de a linha derivada cruzar 0 no ponto mais alto da função faz sentido se você pensar nela logicamente. Se você desenhasse a linha tangente representando a inclinação em cada ponto, estaria girando no sentido horário ao longo do gráfico, apontando inicialmente para cima e para a direita à medida que a bola se ergue e girando até apontar para baixo e para a direita quando a bola cai. No ponto mais alto, a linha tangente seria perfeitamente horizontal, representando uma inclinação de 0.

Execute o seguinte código para visualizar isto.

Exercício 4 - Onde estiver com a palavra RESOLVER, você deve preencher com o código correspondente.

In [11]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def k(x):

    return -10*(x**2) + (100*x)  + 3

def kd(x):

    return -20*x + 100

# Cria uma matriz de valores x para o plot
x = list(range(0, 11))

# Use a função para obter os valores y

y = [k(i) for i in x]

# Use a função derivada para obter as derivadas
# RESOLVER

yd = [kd(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em segundos)')

plt.ylabel('k(x) (altura em centímetros)')

plt.xticks(range(0,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot ds função

plt.plot(x,y, color='green')

# Plot da derivada

plt.plot(x,yd, color='purple')

# Plot das tangentes para x = 2, 5 e 8
x1 = 2

x2 = 5

x3 = 8

plt.plot([x1-1,x1+1],[k(x1)-(kd(x1)),k(x1)+(kd(x1))], color='red')

plt.plot([x2-1,x2+1],[k(x2)-(kd(x2)),k(x2)+(kd(x2))], color='red')

plt.plot([x3-1,x3+1],[k(x3)-(kd(x3)),k(x3)+(kd(x3))], color='red')
```

código p/ construir os tangentes

```
plt.show()
```

Agora considere a seguinte função, que representa o número de flores crescendo em um canteiro de flores antes e depois da pulverização de um fertilizante:

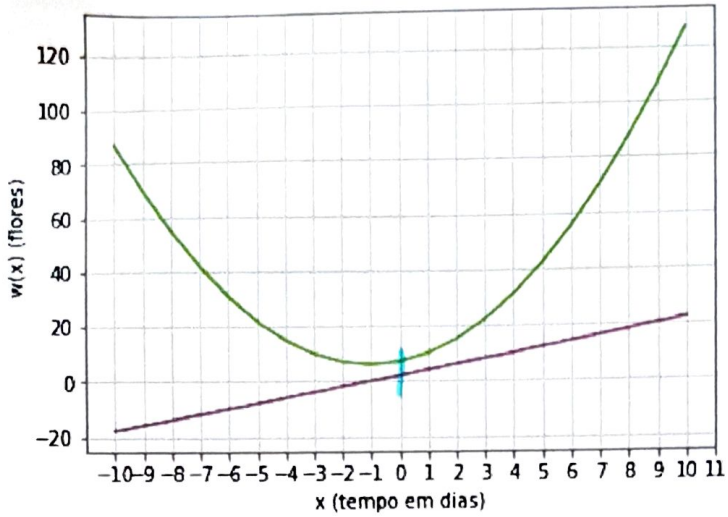
$$\boxed{w(x) = x^2 + 2x + 7}$$

In [12]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def w(x):

    return (x**2) + (2*x) + 7

def wd(x):

    return 2*x + 2
```

derivada da função.

```
# Cria uma matriz de valores x para o plot

x = list(range(-10, 11))

# Use a função para obter os valores y

y = [w(i) for i in x]

# Use a função derivada para obter os valores de y

yd = [wd(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em dias)')

plt.ylabel('w(x) (flores)')

plt.xticks(range(-10,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

# Plot da derivada

plt.plot(x,yd, color='purple')

plt.show()
```



Note que a linha verde representa a função, mostrando o número de flores por 10 dias antes e depois do tratamento com fertilizante. Antes do tratamento, o número de flores estava em declínio e, após o tratamento, o canteiro de flores começou a se recuperar.

A função derivada é mostrada em roxo, e mais uma vez mostra uma mudança linear na inclinação. Desta vez, a inclinação está aumentando a uma taxa constante; e mais uma vez, a linha de função derivada cruza 0 no ponto mais baixo da linha de função (em outras palavras, a inclinação mudou de negativa para positiva quando as flores começaram a se recuperar).

Pontos Críticos

Pelo que vimos até agora, parece que existe uma relação entre uma função que atinge um valor extremo (máximo ou mínimo) e um valor derivativo de 0. Isso faz sentido intuitivo; a derivada representa a inclinação da linha, portanto, quando uma função muda de uma inclinação negativa para uma inclinação positiva, ou vice-versa, a derivada deve passar por 0.

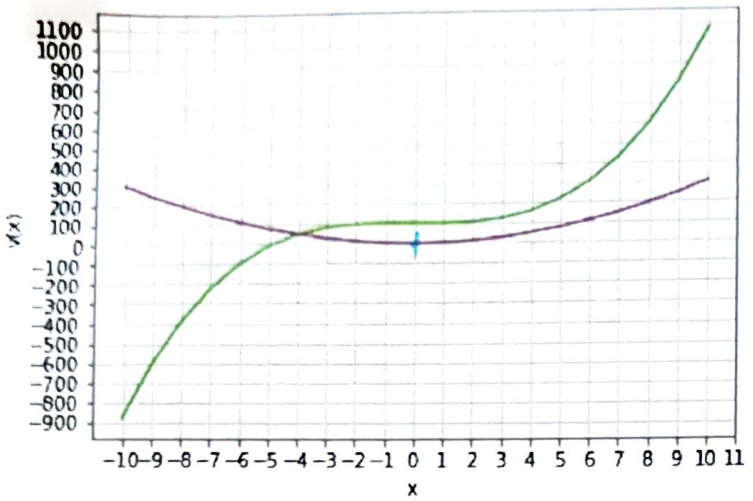
No entanto, você precisa ter cuidado para não assumir que apenas porque a derivada é 0 em um determinado ponto, esse ponto representa o mínimo ou máximo da função. Por exemplo, considere a seguinte função:

$$v(x) = x^3 - 2x + 100$$

Execute o seguinte código Python para visualizar esta função e sua função derivada correspondente.

In [13]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def v(x):

    return (x**3) - (2*x) + 100

def vd(x):

    return 3*(x**2) - 2

# Cria uma matriz de valores x para o plot

x = list(range(-10, 11))

# Use a função para obter os valores y

y = [v(i) for i in x]

# Use a função derivada para obter os valores das derivadas

yd = [vd(i) for i in x]

# Define o gráfico

plt.xlabel('x')

plt.ylabel('v(x)')

plt.xticks(range(-10,15, 1))

plt.yticks(range(-1000, 2000, 100))

plt.grid()

# Plot ds função

plt.plot(x,y, color='green')

# Plot da derivada

plt.plot(x,yd, color='purple')

plt.show()
```



No derivada igual o zero alguma coisa acontece

Observe que, nesse caso, a linha da função derivada roxa passa por 0 à medida que a linha da função verde passa de um declive *côncavo para baixo* (um declive que está diminuindo) para um declive *côncavo para cima* (um declive que aumenta). O declive se estabiliza em 0, formando uma "reta" antes de começar a aumentar.

O que podemos aprender com isso é que coisas interessantes parecem acontecer com a função quando a derivada é 0. Chamamos pontos onde a derivada cruza 0 *pontos críticos* , porque eles indicam que a função está mudando de direção. Quando uma função muda de direção de positiva para negativa, ela forma um pico (ou um *máximo local*), quando a função muda de direção de negativa para positiva forma um vale (ou *mínimo local*) e quando mantém a mesma direção geral, mas altera a concavidade da inclinação, cria um *ponto de inflexão* (saddle).

Encontrando Mínima e Máxima

!

Um uso comum do cálculo é encontrar pontos mínimos e máximos em uma função. Por exemplo, poderíamos querer saber quantos segundos levou para que a bola chutada no jogo de futebol alcançasse sua altura máxima, ou quanto tempo demorou para que nosso fertilizante fosse efetivo em reverter o declínio do crescimento de flores.

Vimos que quando uma função muda de direção para criar um pico máximo ou mínimo, a derivada da função é 0, então um passo em direção a encontrar esses pontos extremos pode ser simplesmente encontrar todos os pontos na função onde a função derivada é 0. Por exemplo, aqui está nossa função para o chute da bola:

$$k(x) = -10x^2 + 100x + 3$$

A partir disso, calculamos a função para a derivada como:

$$k'(x) = -20x + 100$$

Podemos então resolver a equação derivada para um valor f '(x) de 0:

$$-20x + 100 = 0$$

Podemos remover a constante subtraindo 100 a ambos os lados:

$$-20x = -100$$

Multiplicando ambos os lados por -1, elimina os valores negativos (isto não é estritamente necessário, mas torna a equação um pouco menos confusa)

$$20x = 100$$

Assim:

$$x = 5$$

Ponto onde a derivada é zero

Então sabemos que a derivada será 0 quando *x* é 5, mas isso é um mínimo, um máximo ou nenhum dos dois? Poderia ser apenas um ponto de inflexão, ou toda a função poderia ser um valor constante com uma inclinação de 0. Sem olhar para o gráfico, é difícil dizer.

Derivada de Segunda Ordem

A solução para o nosso problema é encontrar a derivada da derivada! Até agora, encontramos a derivada de uma função e a indicamos como ***f'(x)*** . Tecnicamente, isso é conhecido como o derivado *primo* ; e descreve a inclinação da função. Como a função derivada é em si uma função, podemos encontrar sua derivada, que chamamos de derivada de segunda ordem (ou, às vezes, apenas segunda). Isso é indicado assim: ***f ''(x)*** .

Então, aqui está nossa função para o chute da bola:

$$k(x) = -10x^2 + 100x + 3$$

?

f''

Aqui está a função para a derivada principal:

$$k'(x) = -20x + 100$$

E usando uma combinação da regra da potência e a regra de constante, aqui está a função da segunda derivada:

$$k''(x) = -20$$

Agora, sem desenhar o gráfico, podemos ver que a segunda derivada tem um valor constante; então sabemos que a inclinação da derivada principal é linear; e porque é um valor negativo, sabemos que está diminuindo. Assim, quando a derivada principal cruza 0, sabemos que a inclinação da função está diminuindo linearmente; então o ponto em *x = 0* deve ser um ponto máximo.

Execute o seguinte código para traçar a função, a derivada principal e a segunda derivada para o chute da bola.

Exercício 5 - Onde estiver com a palavra RESOLVER, você deve preencher com o código correspondente.

In [14]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def k(x):

    return -10*(x**2) + (100*x)  + 3

# RESOLVER

def kd(x):

    return -20*x + 100

def k2d(x):

    return -20

# Cria uma matriz de valores x para o plot

x = list(range(0, 11))

# Use a função para obter os valores y

y = [k(i) for i in x]

# Use a função derivada para obter os valores de k'(x)

yd = [kd(i) for i in x]

# Use a função da segunda derivada para obter o k'' (x)

y2d = [k2d(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em segundos)')

plt.ylabel('k(x) (altura em centímetros)')

plt.xticks(range(0,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot da função
plt.plot(x,y, color='green')

# Plot  k'(x)
plt.plot(x,yd, color='purple')

# Plot k''(x)
plt.plot(x,y2d, color='blue')

plt.show()
```

Sobe e desce no mesmo proporção

1ª função

2ª função

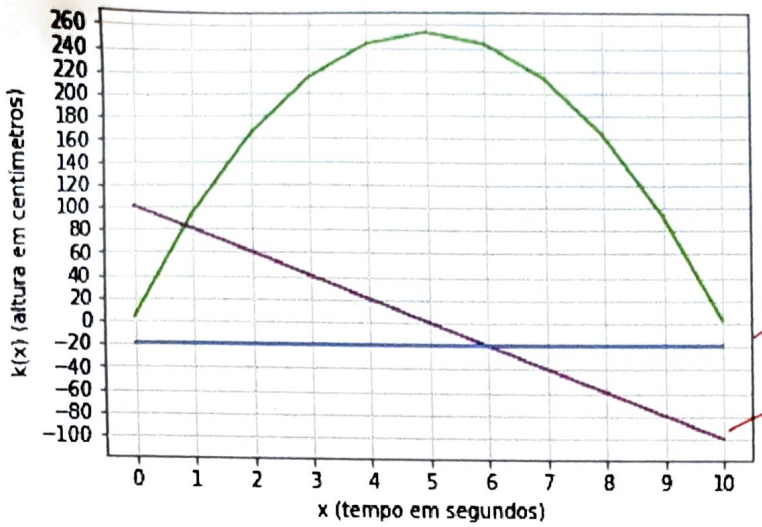
Apenas decrescente

$-20 \cdot x + 100$

3ª função

constante

-20



Vamos tomar a mesma abordagem para o problema do canteiro de flores. Aqui está a função:

$$\boxed{w(x) = x^2 + 2x + 7}$$

Usando a regra da potência e regra constante, nos dá a função derivada principal:

$$\boxed{w'(x) = 2x + 2}$$

A aplicação da regra da potência e regra constante à função derivada principal nos dá a segunda função derivada:

$$\boxed{w''(x) = 2}$$

constante positiva 2

Observe que, desta vez, a segunda derivada é uma constante positiva, portanto, a derivada principal (que é a inclinação da função) está aumentando linearmente. O ponto em que a derivada principal cruza 0 deve, portanto, ser um mínimo. Vamos executar o código abaixo para verificar.

In [15]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Funções
def w(x):

    return (x**2) + (2*x) + 7

def wd(x):

    return 2*x + 2

def w2d(x):

    return 2

# Crie uma matriz de valores x para traçar

x = list(range(-10, 11))

# Use a função para obter os valores y

y = [w(i) for i in x]

# Use a função derivada para obter os valores w'(x)

yd = [wd(i) for i in x]

# Use a função da segunda derivada para obter os valores w '' (x)

y2d = [w2d(i) for i in x]

# Define o gráfico

plt.xlabel('x (tempo em dias)')

plt.ylabel('w(x) (flores)')

plt.xticks(range(-10,15, 1))

plt.yticks(range(-200, 500, 20))

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

# Plot w'(x)

plt.plot(x,yd, color='purple')

# Plot w''(x)

plt.plot(x,y2d, color='blue')

plt.show()
```

87

$-10^2 + 2 \cdot -10 + 7$

$100 + -20 + 7 =$

87

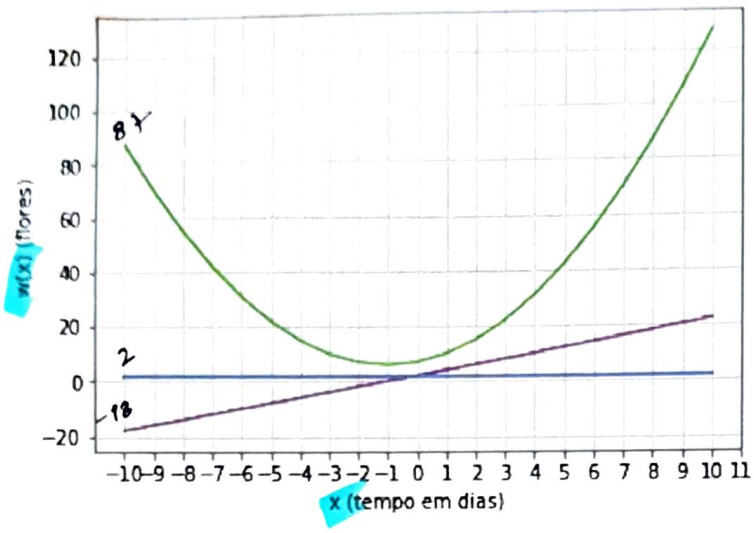
Positivo

-18

$2 \cdot (-10) + 2 = -18$

Positivo

2



Otimização

A capacidade de usar derivadas para encontrar os mínimos e máximos de uma função a torna uma ferramenta útil para cenários em que você precisa otimizar uma função para uma variável específica.

Definindo Funções para Otimização

Por exemplo, suponha que você tenha decidido criar um serviço de vídeo on-line baseado em um modelo de assinatura. Você planeja cobrar uma taxa de assinatura mensal e deseja obter a maior receita possível. O problema é que os clientes são sensíveis ao preço, por isso, se você definir a taxa mensal muito alta, impedirá que alguns clientes se inscrevam. Por outro lado, se você definir a taxa muito baixa, poderá obter mais clientes, mas ao custo da receita reduzida. Problema de encontrar o valor mínimo e máximo

O que você precisa é de algum tipo de função que lhe dirá quantas assinaturas você espera obter com base em uma determinada taxa. Então você fez alguma pesquisa e encontrou uma fórmula para indicar que o volume de assinatura esperado (em milhares) pode ser calculado como 5 vezes a taxa mensal subtraída de 100; ou expresso como uma função:

taxa mensal

$$s(x) = -5x + 100$$

O que você realmente deseja otimizar é a receita mensal, que é simplesmente o número de assinantes multiplicado pela taxa:

?

$$r(x) = s(x) \cdot x$$

Volume de assinatura · x taxa mensal

Nós podemos combinar ***s (x)*** em ***r (x)*** assim:

$$r(x) = -5x^2 + 100x$$

Encontrando a Derivada Principal

A função ***r(x)*** retornará a receita mensal esperada (em milhares) para qualquer taxa proposta (*x*). O que precisamos fazer agora é encontrar a taxa que gera a receita máxima. Felizmente, podemos usar uma derivada para fazer isso.

Primeiro, precisamos determinar a derivada principal de ***r(x)***, e podemos fazer isso facilmente usando a regra da potência:

$$r'(x) = 2 \cdot -5x + 100$$

A qual é:

$$r'(x) = -10x + 100$$

Encontrando Pontos Críticos

Agora precisamos encontrar quaisquer pontos críticos onde a derivada é 0, pois isso pode indicar um máximo:

$$\boxed{-10x + 100 = 0}$$

Vamos isolar o termo *x* :

$$-10x = -100$$

Ambos os lados são negativos, então podemos multiplicar ambos por -1 para torná-los positivos sem afetar a equação:

$$\boxed{10x = 100}$$

Agora podemos dividir ambos os lados por 10 para isolar *x* :

$$x = \frac{100}{10}$$

Assim:

$$\boxed{x = 10}$$

Verificando se há um Máximo

Agora sabemos que com um valor *x* de **10** , a derivada é 0; ou dito de outra forma, quando a taxa é 10, a inclinação indicando a mudança no volume de assinatura é plana. Isso poderia ser um ponto em que a mudança no volume de assinaturas atingiu o pico (em outras palavras, um máximo); mas também pode ser um ponto mínimo ou apenas um ponto de inflexão em que a taxa de mudança muda de negativa para positiva.

Para ter certeza, podemos verificar a derivada de segunda ordem. Podemos calcular isso aplicando a regra da potência à derivada principal:

$$r''(x) = -10$$

Observe que a segunda derivada é uma constante com um valor negativo. Será o mesmo para qualquer ponto, incluindo nosso ponto crítico em *x = 10* :

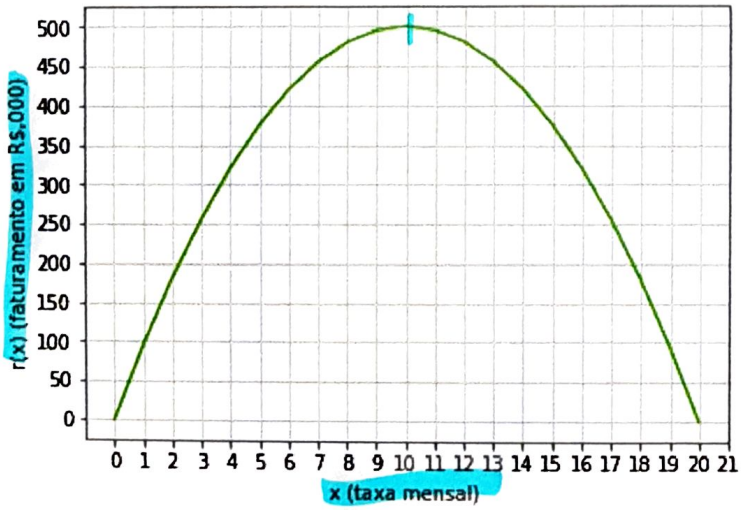
$$r''(10) = -10$$

Um valor negativo para a segunda derivada nos diz que a inclinação derivativa está se movendo em uma direção negativa no ponto em que é 0, portanto, o valor da função deve estar no máximo.

Em outras palavras, a taxa mensal ideal para o nosso serviço de vídeo on-line é 10 - isso gerará a receita mensal máxima.

In [16]:

```
from matplotlib import pyplot as plt
%matplotlib inline

# Função s
def s(x):

    return (-5*x) + 100

# Função r
def r(x):

    return s(x) * x

# Cria uma matriz de valores x para o plot

x = list(range(0, 21))

# Use a função para obter os valores y

y = [r(i) for i in x]

# Define o gráfico

plt.xlabel('x (taxa mensal)')

plt.ylabel('r(x) (faturamento em R$,000)')

plt.xticks(range(0,22, 1))

plt.yticks(range(0, 600, 50))

plt.grid()

# Plot da função

plt.plot(x,y, color='green')

plt.show()
```



Cálculo Integral

Integrais são os inversos de derivadas. Mais importante, o uso da integração fornece uma maneira de calcular a área sob a curva de praticamente qualquer função. Existem muitas aplicações para integração. Por exemplo, se você precisar calcular uma probabilidade de alguma ocorrência entre limites (que discutiremos mais adiante neste curso), você usará uma integral.

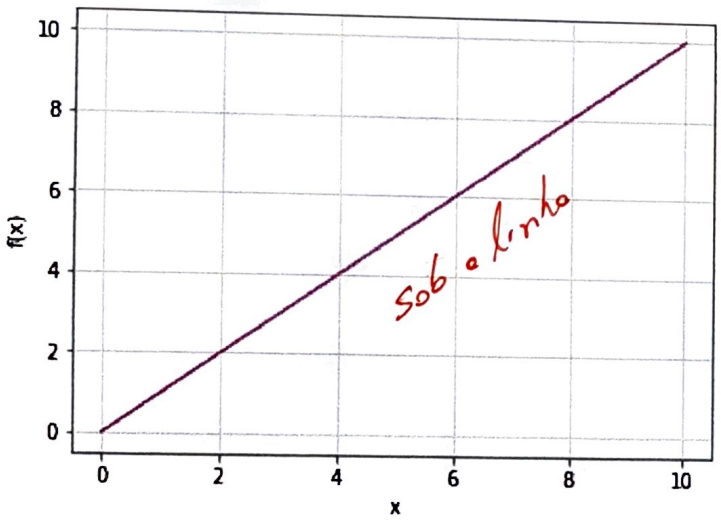
Vamos começar com uma função simples:

$$f(x) = x$$

Podemos traçar essa função como uma linha. Execute o código abaixo para traçar a função para o intervalo de 0 a 10.

In [17]:

```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

# Função
def f(x):
    return x

# Cria uma matriz de valores x de 0 a 10

x = range(0, 11)

# Obtém os valores y correspondentes da função
y = [f(a) for a in x]

# Define o gráfico

plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot x e f(x)

plt.plot(x,y, color='purple')


plt.show()
```



Executando Integração

O *integral* de uma função é a área sob ela - nesse caso, a área sob a linha diagonal roxa até o eixo x.

Então, como você encontra a integral de uma função? bem, para nossa função simples $f(x) = x$, a fórmula para uma integral é escrita da seguinte forma:

$$\int f(x)\,dx$$

O símbolo $\int$ mostra que esta fórmula é uma integral. O dx indica que a integração é em relação à variável x. Observe que, como $f(x) = x$, também poderíamos escrever essa fórmula integral como $\int x\,dx$

Então, qual é a integral de $x\,dx$? Para responder a essa pergunta, precisamos da *antiderivada* de **f** - em outras palavras, precisamos encontrar uma função que tenha uma derivada que corresponda à saída de f, que é apenas x. Usando a regra da potência no sentido inverso, uma função que tenha a derivada x seria $\frac{1}{2}x^2$

Então, a fórmula da integral para f em relação a x pode ser escrita como:

$$\boxed{\int f(x)\,dx = \frac{1}{2}x^2}$$

Integração entre limites

Agora que temos a fórmula integral, podemos usá-la para encontrar a integral entre pontos iniciais e finais específicos. Vamos supor que queremos encontrar a área sob a função entre os valores x 0 e 2. Em outras palavras, o *integral* de f para o intervalo de 0 a 2 em relação a x.

Execute o seguinte código para reformular a função e mostrar a área em que estamos interessados:

$$\sqrt[3]{x} = x^{\frac{1}{3}}$$

$$\frac{1}{3} + 1 = \frac{1}{3} + \frac{3}{3} = \frac{4}{3}$$

$$\frac{1}{\frac{4}{3}} = \frac{3}{4}$$

$\frac{1}{3}$ +

$\frac{3}{3}$

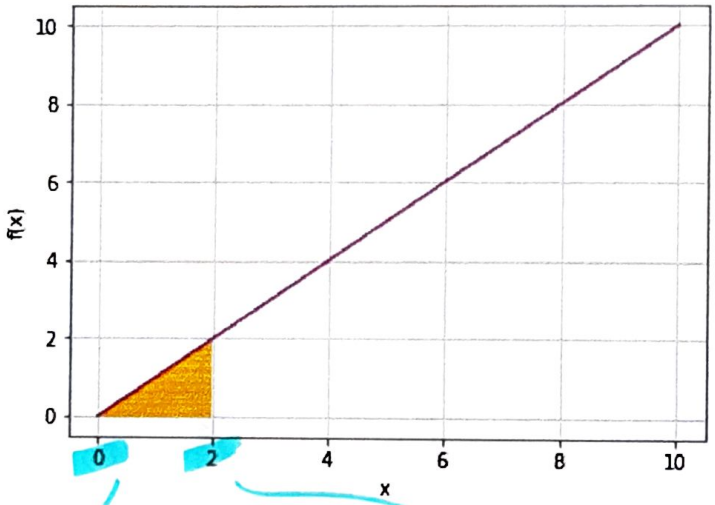
$= \frac{4}{3}$

$$\sqrt[3]{x^4} = \sqrt[\cancel{3}]{x^{\cancel{3}} \cdot x} = x \cdot \sqrt[3]{x}$$

In [18]:

```
# Define o gráfico
plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()


# Plot x e f(x)

plt.plot(x,y, color='purple')

# Mostra a área para a integral

section = np.arange(0, 2, 1/20)

plt.fill_between(section,f(section), color='orange')

plt.show()
```

está usando a função instanciada em outra célula



Chamamos o ponto inicial e final os **limites da integral**. O limite inferior é colocado como um subscrito do sinal integral. O limite superior é colocado como um sobrescrito do sinal integral. Usando essa notação, a integral de $f(x)$ de 0 para 2 é escrita da seguinte maneira:

$$\boxed{\int_0^2 f(x)\,dx}$$

A integral é avaliada subtraindo-se o valor do integrando no limite inferior do integrante no limite superior; e como sabemos a fórmula baseada em nossa função antiderivada, a integral pode ser avaliada da seguinte maneira.

$$\int_0^2 f(x)\,dx$$
$$= \frac{1}{2}x^2 \Big|_0^2$$
$$= \frac{1}{2}2^2 - \frac{1}{2}0^2$$
$$= \frac{4}{2} - \frac{0}{2}x^2$$
$$= 2$$

?

Exercício 6 - Crie o código em Python e verifique se o resultado retornado pela função *scipy.integrate.quad* é aproximadamente o mesmo que calculamos analiticamente.

In [19]:

```
# Solução
import scipy.integrate as integrate

i, e = integrate.quad(lambda x: f(x), 0, 2)

print (i)
```

função anônima

calcula integral em python

```
2.0
```

Retorna integral e taxa de erro

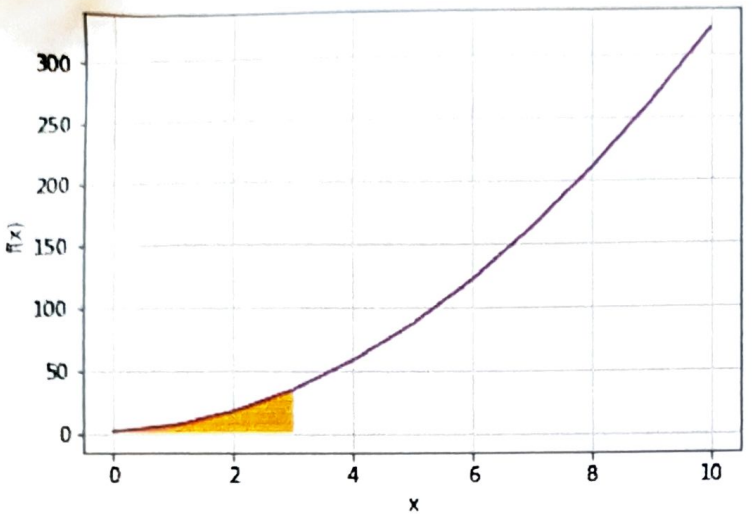
Outra Integral

Aqui está outro exemplo para uma função um pouco mais complexa. Qual é a área sob a curva da função $3x^2 + 2x + 1$ entre 0 e 3?

Vamos olhar para essa função e a área em questão:

In [20]:

```
import numpy as np

from matplotlib import pyplot as plt

from matplotlib.patches import Polygon

%matplotlib inline

# Função
def g(x):

    return 3 * x**2 + 2 * x + 1

# Cria uma matriz de valores x de 0 a 10

x = range(0, 11)

# Obtém os valores y correspondentes da função

y = [g(a) for a in x]

# Define o gráfico

fig, ax = plt.subplots()

plt.xlabel('x')

plt.ylabel('f(x)')

plt.grid()

# Plot x e g(x)

plt.plot(x,y, color='purple')

# Desenha a região sombreada
ix = np.linspace(0, 3)

iy = g(ix)

verts = [(0, 0)] + list(zip(ix, iy)) + [(3, 0)]

poly = Polygon(verts, facecolor='orange')

ax.add_patch(poly)

plt.show()
```



Podemos avaliar esta integral como antes, desta vez usando a função:

$$\int_0^3 3x^2 + 2x + 1 \; dx$$

Podemos calcular a antiderivada de $3x^2 + 2x + 1 \; dx$ como $\frac{3}{3}x^3 + \frac{2}{2}x^2 + x$, portanto:

$$\int_0^3 = \frac{3}{3}x^3 + \frac{2}{2}x^2 + x \Big|_0^3$$
$$= \frac{3}{3}3^3 + \frac{2}{2}3^2 + 3 - \frac{3}{3}0^3 - \frac{2}{2}0^2 + 0$$
$$= 27 + 9 + 3 + 0 + 0 + 0$$
$$= 39$$

Exercício 7 - Agora, crie o código em Python para verificar o resultado.

In [21]:

```
# Solução
i, e = integrate.quad(lambda x: 3 * x**2 + 2 * x + 1, 0, 3)
print(i)
```

```
38.99999999999999
```

Note que o resultado da função ***scipy.integrate.quad*** é aproximado - a função realmente retorna uma integral estimada (*i* neste caso) e também uma medida de erro absoluto (*e*).

Exercício 8 - Qual o código Python para ver qual foi o erro absoluto nesse caso?

In [22]:

```
# Solução
print(e)
```

```
4.3298697960381095e-13
```

O erro absoluto neste caso é extremamente pequeno (aproximadamente 4.3×10^{-13}).

Limites infinitos

Em muitos casos, os limites de uma integral podem ser $+/-\infty$. Talvez surpreendentemente, esta situação não é realmente um problema se a função que está sendo integrada converge para 0 no limite infinito.

Aqui está um exemplo. A função $e^{-5x} \rightarrow 0$ como $x \rightarrow \infty$. Portanto, a integral desta função de algum limite para ∞. Esta integral pode ser escrita da seguinte maneira:

$$\int_0^\infty e^{-5x} dx$$

Exercício 9 - Calcule a integral numericamente em Python.

In [23]:

```
# Solução
import numpy as np
i, e = integrate.quad(lambda x: np.exp(-x*5), 0, np.inf)

print('Integral: ' + str(i))
print('Erro Absoluto: ' + str(e))
```

```
Integral: 0.20000000000000004
Erro Absoluto: 1.5606666951908062e-11
```

Essa integral converge para um pequeno número com uma estimativa de erro muito menor.

Aqui está outro exemplo que ilustra porque ter limites infinitos de integração é tão útil. Ao computar probabilidades, muitas vezes é necessário ter limites infinitos. Não se preocupe muito com os detalhes da teoria da probabilidade. Isso é coberto em uma lição posterior.

Uma distribuição Normal com média zero e um desvio padrão de 1 tem a seguinte função de densidade:

$$\frac{1}{2\pi} e^{\frac{-x^2}{\sqrt{(2\pi)}}}$$

Faz sentido que a integral dessa função de densidade de probabilidade de $-infty$ para $infty$ deva ser de 1.0. Em outras palavras, a probabilidade de um evento normalmente distribuído ocorrer em todos os valores possíveis deve ser de $1,0$.

O código na célula abaixo calcula a seguinte integral:

$$\int_{-\infty}^{\infty} \frac{1}{2\pi} e^{\frac{-x^2}{\sqrt{(2\pi)}}} dx$$

Exercício 10 - Calcule a integral numericamente em Python e verifique se o resultado é aproximadamente 1.0.

In [24]:

```
# Solução
import numpy as np
norms = lambda x: np.exp(-x**2/2.0)/np.sqrt(2.0 * 3.14159)
i, e = integrate.quad(norms, -np.inf, np.inf)

print('Integral: ' + str(i))
print('Erro Absoluto: ' + str(e))
```

```
Integral: 1.0000004223321999
Erro Absoluto: 1.017819568483304e-08
```

In [ ]:

```

```

Fim