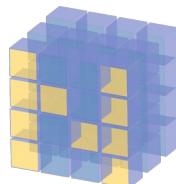
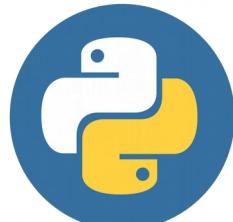


# Mini Curso de Programação em Python

--*João Victor Vilela Cassiano*--

--e-mail : [joaocassianox7x@gmail.com](mailto:joaocassianox7x@gmail.com)

--repositório : <https://github.com/joaocassianox7x>



NumPy



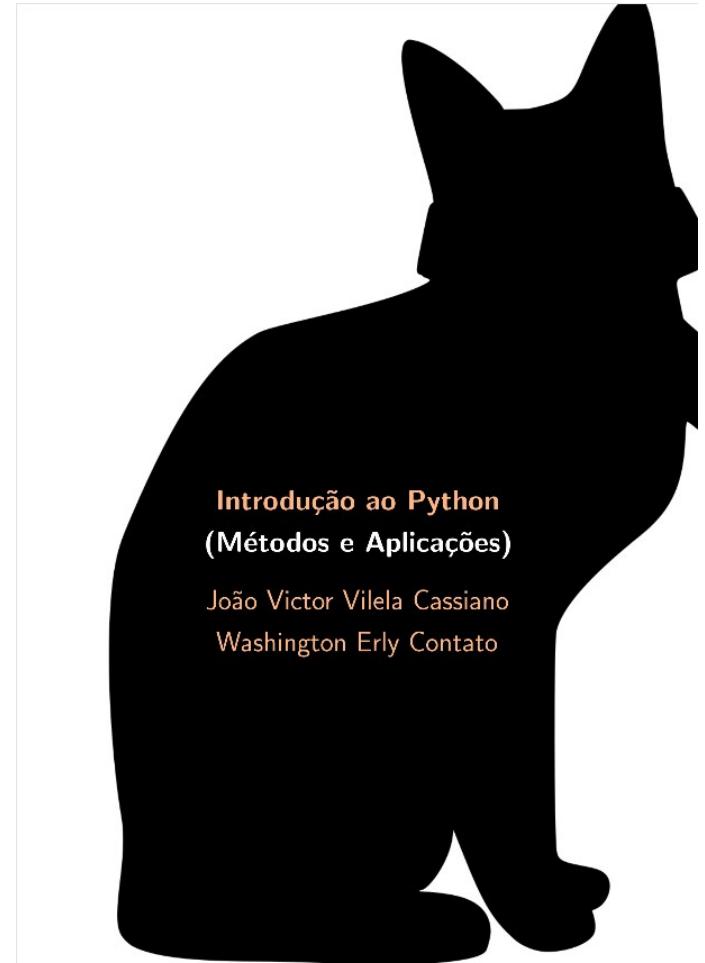
TensorFlow



Keras

# Recomendações:

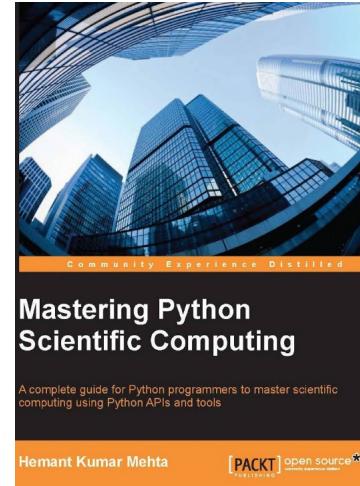
- Sou aluno da Física de Materiais na UFU e se Deus quiser formo ao final do ano. Minha pequena contribuição à comunidade foi uma apostila de introdução ao Python que, modéstia a parte é muito boa (Apostila). Trata-se de uma apostila em português dividida em três capítulos: 1º é uma introdução ao python (~17págs); o 2º fala sobre algorítmos para solução de EDO's, Transformada de Fourier e Integração Numérica; já o 3º ensino a usar o NumPy, Matplotlib e o SciPy.



# Recomendações:

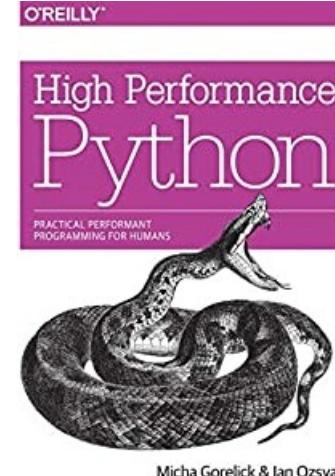
## Iniciantes:

- *Mastering Python Scientific Computing*;
- *Minha Apostila*;
- *Numerical Methods With Fortran IV Case Studies*;
- *Entendendo Algoritmos: Um Guia Ilustrado; Para Programadores e Outros Curiosos*;
- *Curso Intensivo de Python*.



## Não Iniciantes:

- *Usin OpenMP*;
- *High Performance Python*;
- *Applied Numerical Linear Algebra*;
- *Performance Optimization, Numerically Intensive Codes*;
- *Problemas Clássicos de Ciência da Computação com Python*.



# Sumário do Mini Curso:

- 1.Uma breve revisão;
- 2.Matplotlib → Visualização de Dados;
3. NumPy → Pacote Numérico;
4. Redes Neurais → Melhor pacote front-end e alguns comentários;

O curso com os exemplos pode ser encontrado em : [Repositório](#)

The screenshot shows a GitHub repository page for the user 'joacassianox7x'. The repository has 1 branch and 0 tags. The README.md file contains the following text:

```
MINICURSO_PET

Repositório com programas e exemplos mostrados no minicurso oferecido pelo PET-Física da Universidade Federal de Uberlândia.

Sinta-se livre para compartilhar o material, quaisquer comentários ou dúvidas terei prazer em responder via e-mail (joacassianox7x@gmail.com)
```

The repository has 2 commits, the latest being an initial commit by 'joacassianox7x' 14 minutes ago. The repository also includes a .gitignore file and a README.md file.

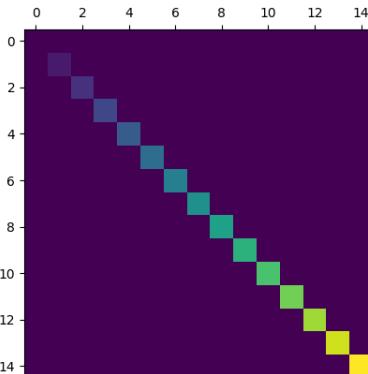
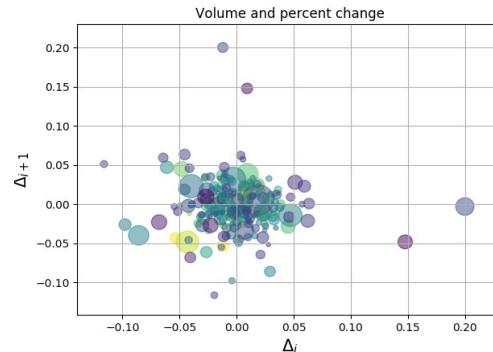
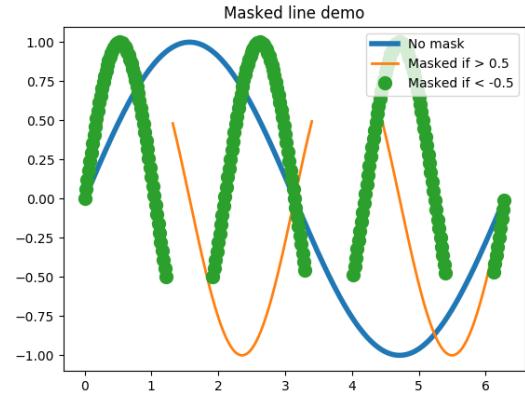
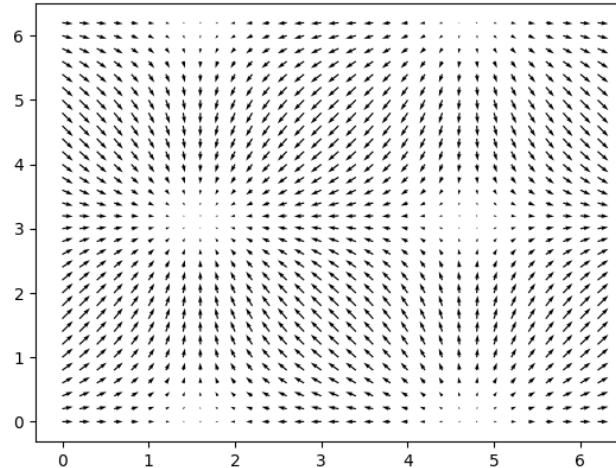
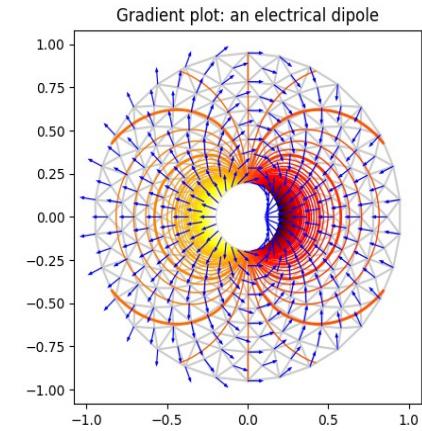
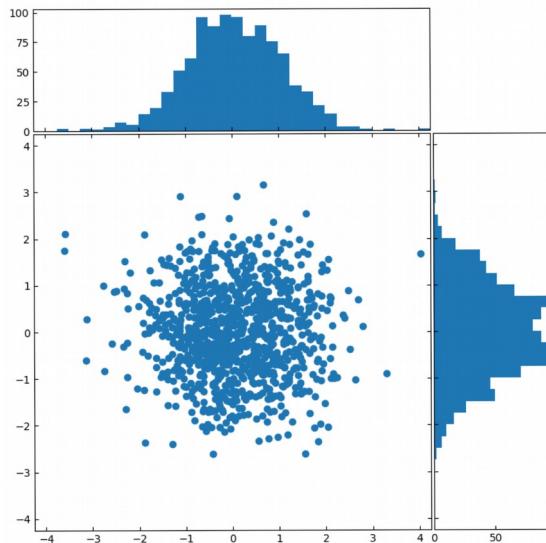
**About**  
No description, website, or topics provided.  
[Readme](#)

**Releases**  
No releases published  
[Create a new release](#)

**Packages**  
No packages published  
[Publish your first package](#)

# • Matplotlib

- O Matplotlib é um pacote extremamente robusto, felizmente possui uma das melhores documentações ([doc](#) ).
- O código é todo escrito sobre o Numpy, e possibilita a criação de figura e vídeos extremamente bonitos.



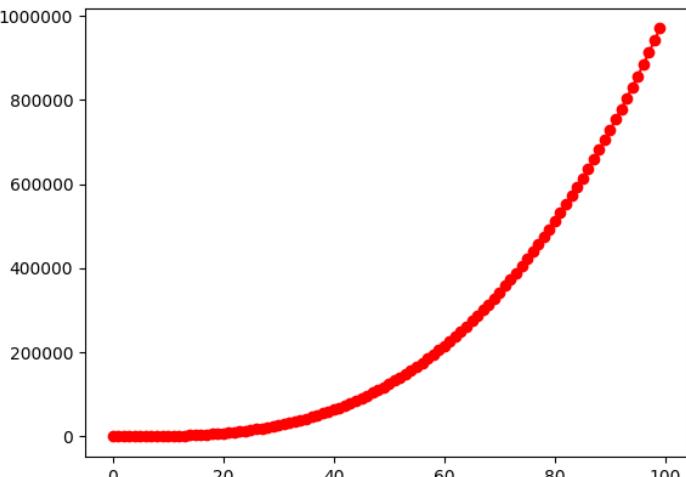
- No matplotlib o pacote que você com certeza mais vai utilizar é o pyplot. Vamos trabalhar e discutir um pouco sobre ele aqui.

```
import matplotlib.pyplot as plt
```

```
x=range(100)
y=[i**2-1 for i in x]
```

fmt='o-r'

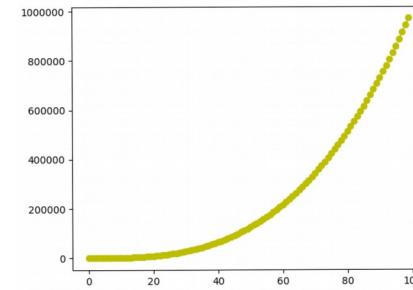
plt.plot(x,y,fmt)



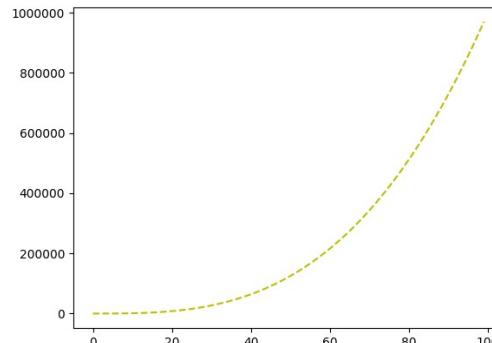
fmt : indica o formato do plot, “o” para bolinhas, “-” para traço entre as bolinhas e “r” para que o plot seja vermelho.

- Na documentação do matplotlib, você encontra todas as formatações possíveis para o plot

**fmt='o--y'**



**fmt='--y'**

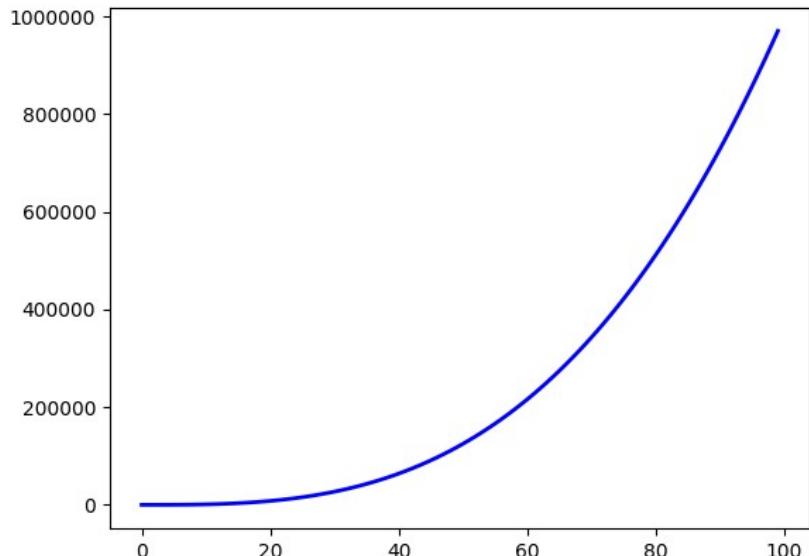


- Podemos ainda adicionar no comando plot a espessura da linha/pontilhado:

```
import matplotlib.pyplot as plt
```

```
x=range(0,100)  
y=[k**3 for k in x]
```

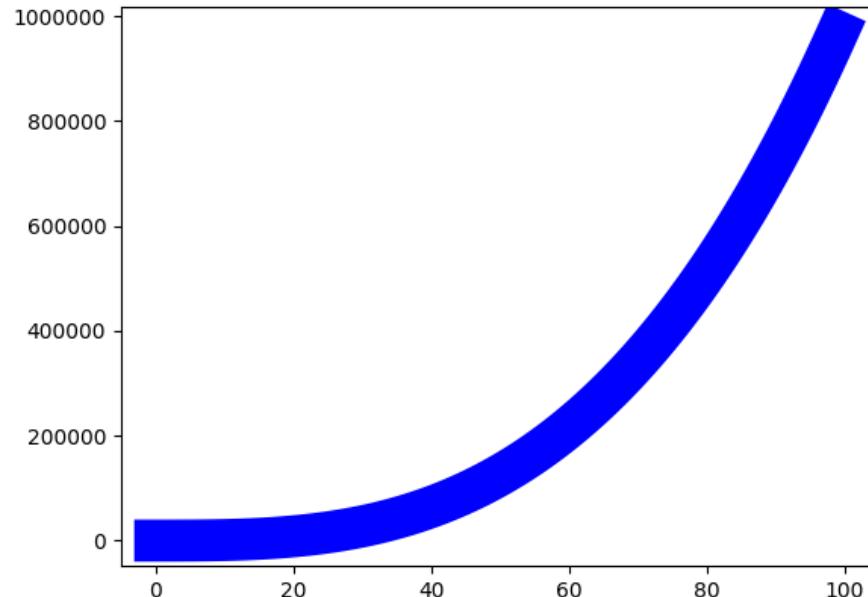
```
plt.plot(x,y,'-b',lw=2)
```



```
import matplotlib.pyplot as plt
```

```
x=range(0,100)  
y=[k**3 for k in x]
```

```
plt.plot(x,y,'-b',lw=20)
```



- Vamos incrementar um pouco mais nosso código, com algumas coisas básicas e extremamente importantes:

```
import matplotlib.pyplot as plt
```

```
x=range(0,100)
y=[k**3 for k in x]
```

```
plt.rcParams.update({'font.size': 16})
plt.title('Funcao')
plt.grid(True)
plt.xlim(x[0],x[-1])
plt.ylim(x[-1]**3)
plt.plot(x,y,'-b',lw=2)
plt.legend([R'$f(x)=x^3$'],loc='upper right')
```

```
plt.tight_layout()
plt.savefig('nome.png')
```

mudamos a fonte global

adiciona um título

adiciona grade

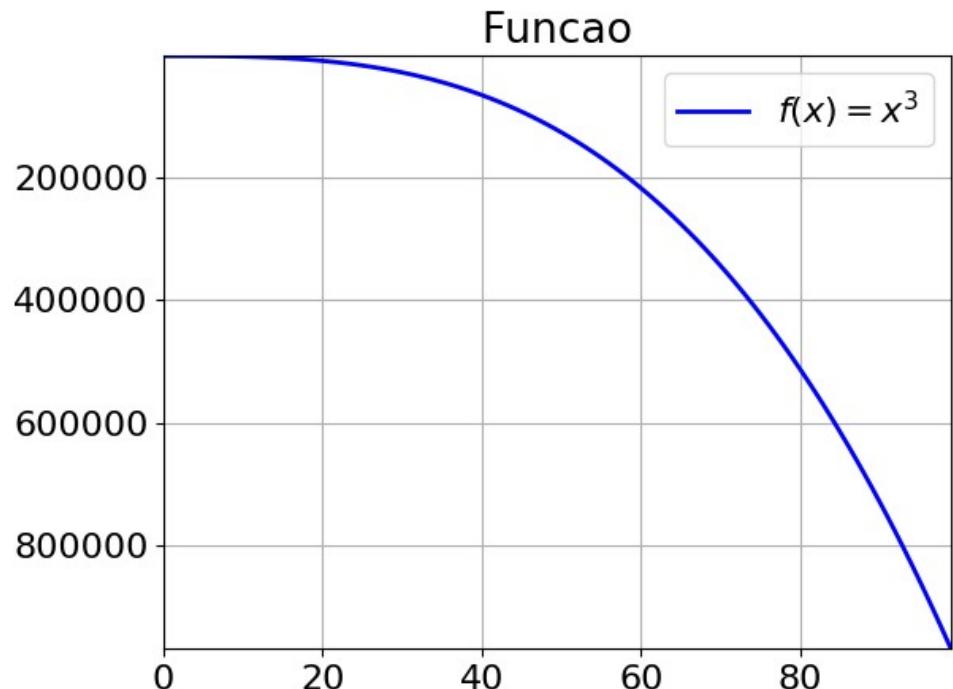
limite do eixo x

limite em y

legenda no canto superior direito

evita que a imagem se disforme ao ser salva

salvar a imagem como .png



E se quisermos vários plots na mesma figura?  
Basta escrevermos vários plots:

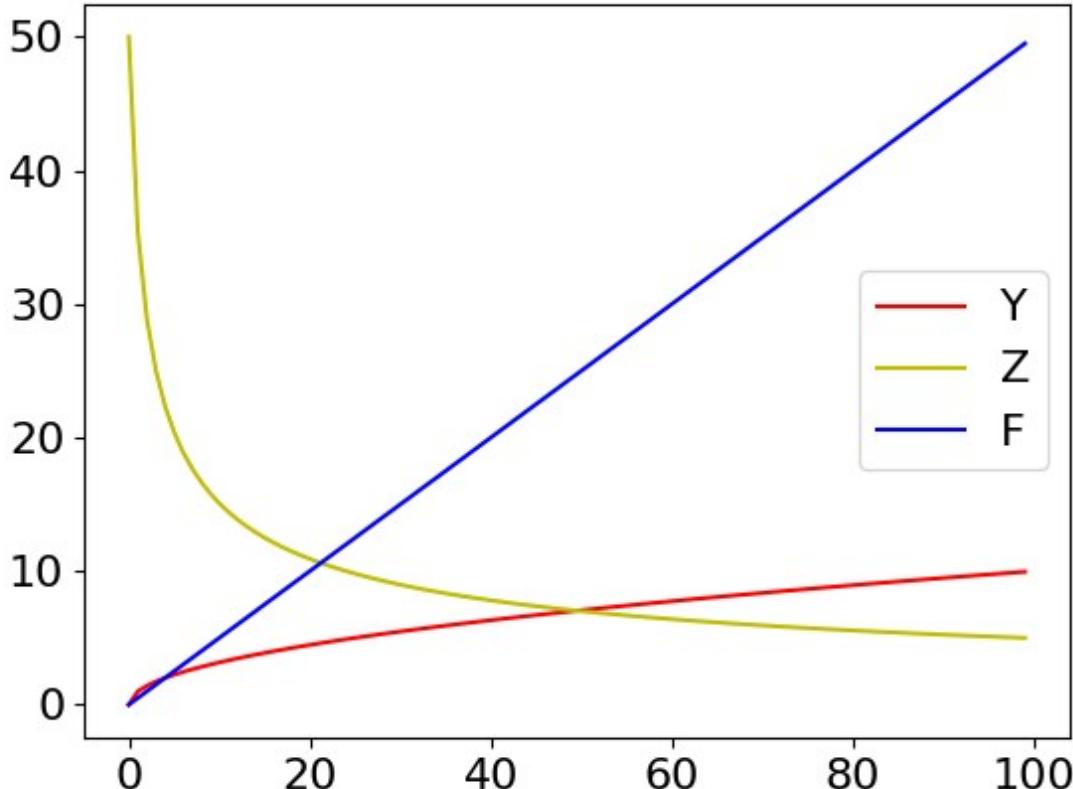
```
import matplotlib.pyplot as plt

x=range(100)
y=[i**0.5 for i in x]
z=[50/(i+1)**0.5 for i in x]
f=[i/2 for i in x]

plt.plot(x,y,'r',lw=1.5)
plt.plot(x,z,'y',lw=3)
plt.plot(x,f,'b',lw=5)

plt.legend(['Y','Z','F'],loc='best')

plt.tight_layout()
plt.savefig('nome.png')
```

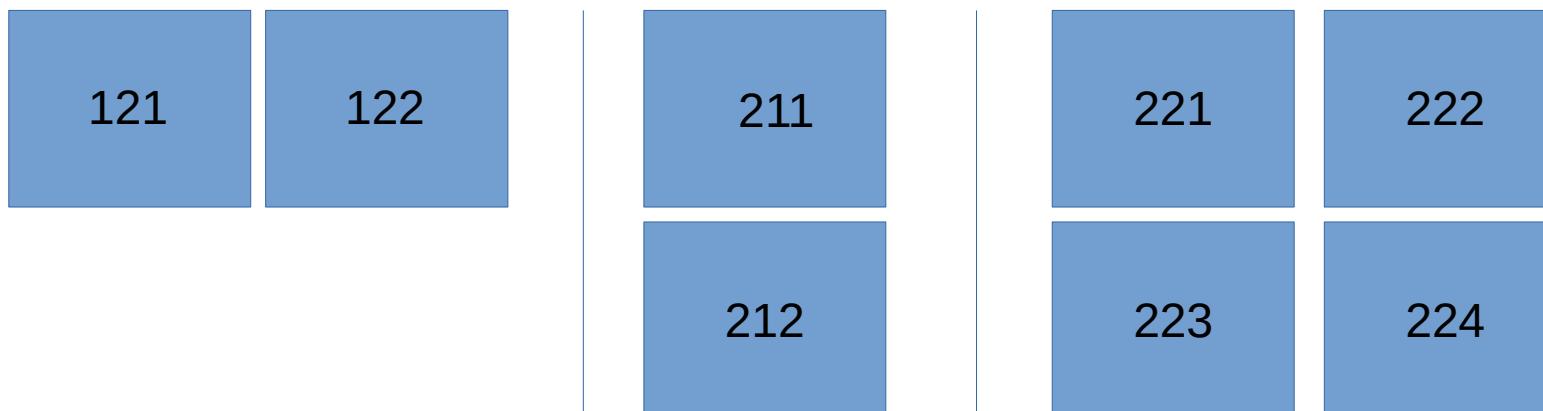


- Agora, como colocar vários gráficos no mesmo programa, mas cada um habitando uma figura diferente?
- Utilizamos o comando “plt.subplot”

```
plt.subplot(211)  
plt.plot(x,y)
```

```
plt.subplot(212)  
plt.plot(x,k)
```

- O código acima diz que haverá duas linhas e uma coluna e, “x,y” como a primeira figura, “x,k” a segunda. Por exemplo:



- Por exemplo:

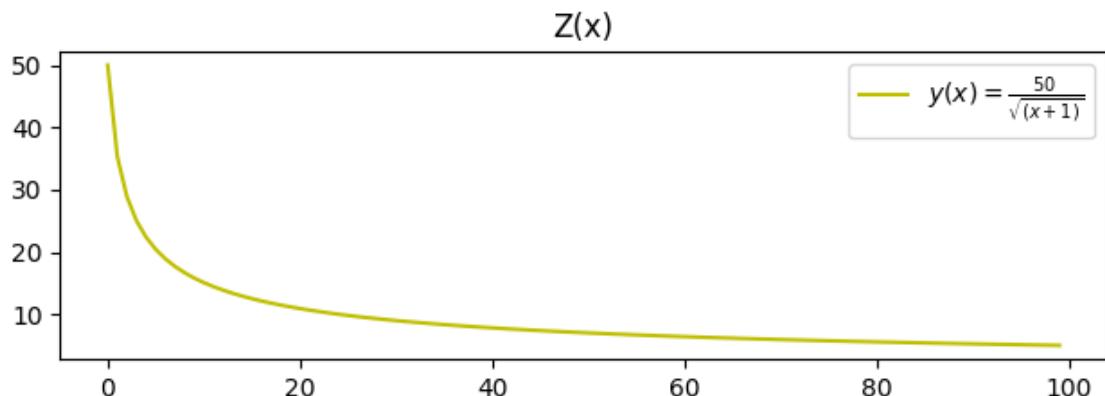
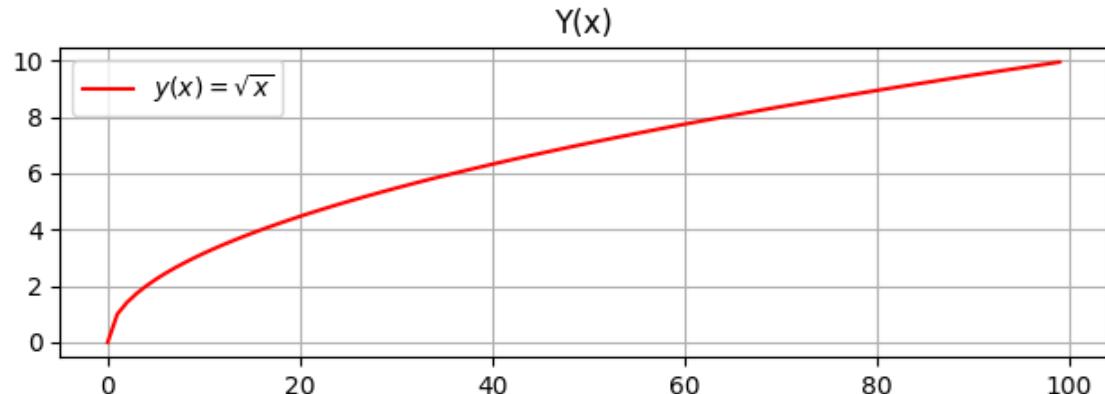
```
import matplotlib.pyplot as plt
import matplotlib
x = range(100)
y=[x[i]**0.5 for i in range(len(x))]
z=50/(x+1)**0.5[50/(x[i]+1)**0.5 for i in range(100)]

matplotlib.rcParams.update({'font.size': 10})

plt.subplot(211)
plt.title('Y(x)')
plt.plot(x,y,'r')
plt.legend([R'$y(x)=\sqrt{x}$'],loc='upper left')
plt.grid(True)

plt.subplot(212)
plt.title('Z(x)')
plt.plot(x,z,'y')
plt.legend([R'$y(x)=\frac{50}{\sqrt{(x+1)}}$'],loc='upper right')
plt.grid(False)

plt.tight_layout()
```



## • Numpy

- Numpy é o pacote numérico de Python, extremamente rápido e muito otimizado. No geral, sempre utilizaremos numpy, em 90% dos programas, esse será seu primeiro pacote importado;
- E por que é bom? Possui ferramentas rápidas para trabalharmos com álgebra, o linalg (usufrui do Lapack), FFT (fast fourier transform) e DFT (diferencial fourier transform) que usufruem dos respectivos pacotes em “C”, Random para dados aleatórios;

- Numpy é tão importante, que o curso inteiro poderia ser sobre como utilizar o Numpy para o maior número de tarefas possível. Muitos dos pacotes científicos são escritos sobre o Numpy, como: Scipy, Matplotlib, Keras, TensorFlow, Pandas, Seaborn e etc...



Espero muito que Monte Carlo seja de conhecimento geral, mas teremos uma breve discussão, sem problemas...

$10^6$  interações

Monte Carlo sem Numpy: 1.77

Monte Carlo com Numpy: 0.04

## • Numpy vs Python

- O numpy possui uma estrutura mais simples e prática para se trabalhar com vetores, embora seja importante saber escrever funções puras (sem pacotes) otimizadas no geral utilizaremos o Numpy para quase tudo.
- O Numpy possui como base as “arrays”, que se diferem muito das listas, por exemplo:

```
import numpy as np
```

```
I=[1,2,3]
```

```
n=np.array[1,2,3]
```

I+I → [1,2,3,1,2,3]

I+n → retorna um objeto array [2,4,6]

I\*\*2 → erro

n\*\*2 → [1,4,9]

np.sin(n) → transforma “n” em array e [sin(1),sin(2),sin(3)]

np.exp(I) → [sin(1),sin(2),sin(3)]

## • Comandos úteis:

```
import numpy as np
```

```
pi=np.pi
```

```
e=np.e
```

```
I=np.arange(a,b,c)
```

```
j=np.linspace(d,e,f)
```

```
k=np.zeros((a,b,c,...))
```

cria uma array que vai de a até b-c de c em c

uma array que vai de a até e em f pontos

Um tensor de ordem “n” com a, b,c,... espaços reservados para cada índice.

## Exemplo:

- Crie uma matriz onde a primeira coluna sejam valores de seno e, a segunda coluna valores de cosseno variando de zero a pi.

```
import numpy as np
```

```
m=100 #numero de pontos
```

```
pi=np.pi
```

```
x=np.range(0,2*pi,m)
```

```
I1=np.cos(x)
```

```
I2=np.sin(x)
```

```
m=np.zeros((m,2))
```

```
m[:,0]=I1
```

```
m[:,1]=I2
```

```
print(m)
```

- Ok, agora façamos algo um pouco mais “complexo”. Plete quatro gráficos na mesma figura, para quatro funções diferentes.

```
import numpy as np
```

```
m=100 #numero de pontos
```

```
pi=np.pi
```

```
x=np.range(0,2*pi,m)
```

```
I1=np.cos(x)
```

```
I2=np.sin(x)
```

```
m=np.zeros((m,2))
```

```
m[:,0]=I1
```

```
m[:,1]=I2
```

```
print(m)
```

- Ok, agora façamos algo um pouco mais “complexo”. Plete quatro gráficos na mesma figura, para quatro funções diferentes.

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

```
x=np.arange(0,10+0.01,0.01)
```

```
y1=np.exp(x)
```

```
y2=np.exp(-x**2)
```

```
y3=np.sin(x)**3
```

```
y4=np.cos(x)**2
```

```
plt.subplot(4,4,1)  
plt.plot(x,y1)
```

```
plt.subplot(4,4,2)  
plt.plot(x,y2)
```

```
plt.subplot(4,4,3)  
plt.plot(x,y3)
```

```
plt.subplot(4,4,4)  
plt.plot(x,y4)
```

```
plt.legend([R'$e^x$', R'$e^{-x^2}$',  
          R'$\sin(x)$', R'$\cos(x)$'])
```

```
plt.tight_layout()  
plt.savefig('exemplos.png')
```

- **Linalg**
- O linalg (linear algebra package) é pacote de álgebra linear do Numpy. Aqui são feitas algumas operações básicas, mas essenciais como calcular auto-vetor e auto-valor, calcular determinante, diagonalizar matriz, produto entre matrizes, mudança de base, encontrar inversa e etc...

```
import numpy as np
import numpy.linalg as alg
```

```
m=np.array([[1j,0],[0,-1j]])
```

```
ava,ave=alg.eigh(m)
det=alg.det(m)
inv=alg.inv(m)
```

inversa de m

auto valores como vetor  
e auto vetores como  
coluna das matrizes

determinante de m

Depois, escreva uma matriz cuja diagonal principal seja “2” e as diagonais inferior e superior “-1”, encontre e plote o módulo quadrado dos seus auto-vetores.

$$M = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

- **Linalg**

- O linalg (linear algebra package) é pacote de álgebra linear do Numpy. Aqui são feitas algumas operações básicas, mas essenciais como calcular auto-vetor e auto-valor, calcular determinante, diagonalizar matriz, produto entre matrizes, mudança de base, encontrar inversa e etc...

```
import numpy as np  
import numpy.linalg as alg
```

```
m=np.array([[1j,0],[0,-1j]])
```

```
ava,ave=alg.eigh(m)
```

```
det=alg.det(m)
```

```
inv=alg.inv(m)
```

inversa de m

auto valores como vetor  
e auto vetores como  
coluna das matrizes

determinante de m

Depois, escreva uma matriz cuja diagonal principal seja “2” e as diagonais inferior e superior “-1”, encontre e plote o módulo quadrado dos seus auto-vetores.

$$M = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

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

```
m=2*np.eye(ordem,k=0)  
m+=-1*np.eye(ordem,k=1)  
m+=-1*np.eye(ordem,k=-1)
```

```
val, vec=alg.eigh(m)
```

```
y1, y2,  
y3=np.abs(vec[:,0])**2,np.abs(vec[:,1])**2,np.  
abs(vec[:,2])**2
```

```
plt.plot(y1)  
plt.plot(y2)  
plt.plot(y3)
```

- Como álgebra é um negócio útil a todos, façamos mais alguns exemplos!

```
import numpy.linalg as alg  
import numpy as np
```

```
dim = 100  
Me = np.random.random(dim, dim) #matriz aleatória  
Me += np.transpose(Me) #matriz hermitiana
```

```
traco = np.trace(Me) #traco da matriz  
norma = alg.norm(Me) #norma da matriz/vetor
```

```
b = np.random.random(dim) #gera um vetor aleatório  
x = alg.solve(Me,b) #resolve problema do tipo A.x = b
```

- Mas um problema é especialmente importante na álgebra linear é o problema de auto-valores e auto-vetores. Para isso temos várias sub-rotinas:

```
import numpy.linalg as alg  
import numpy as np
```

```
dim = 100 #dimensao da matriz  
Me = np.random.random((dim, dim))  
Me+=np.transpose(Me)
```

```
Ma = np.random.random((dim, dim))  
#matriz aleatória
```

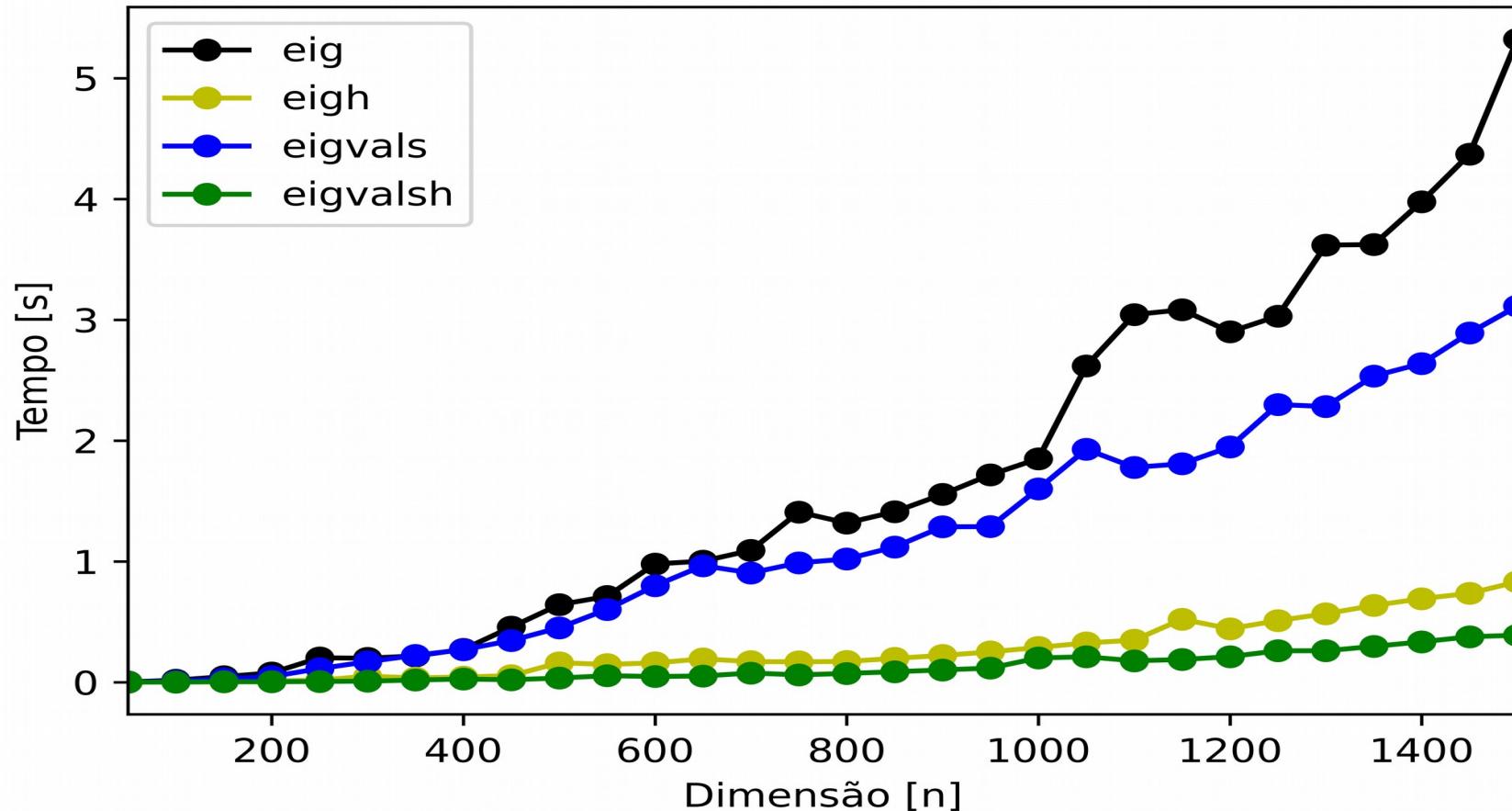
```
val1,vel1 = alg.eig(Ma) #autovalores e  
autovetores de uma matriz qualquer
```

```
vec2,vel2 = alg.eigh(Me) #autovalores e  
autovetores de uma matriz hermitiana
```

```
val1 = alg.eigvals(Ma) #autovalores de  
uma matriz qualquer
```

```
val2 = alg.eigvalsh(Me) #autovalores de  
uma matriz hermitiana
```

Por que é importante saber dessas funções? Vamos discutir sobre o gráfico abaixo!



- Agora que você já conhece NumPy e Matplotlib, façamos alguns plots mais complexos, que tal? Primeiro, um histograma em duas dimensões:

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

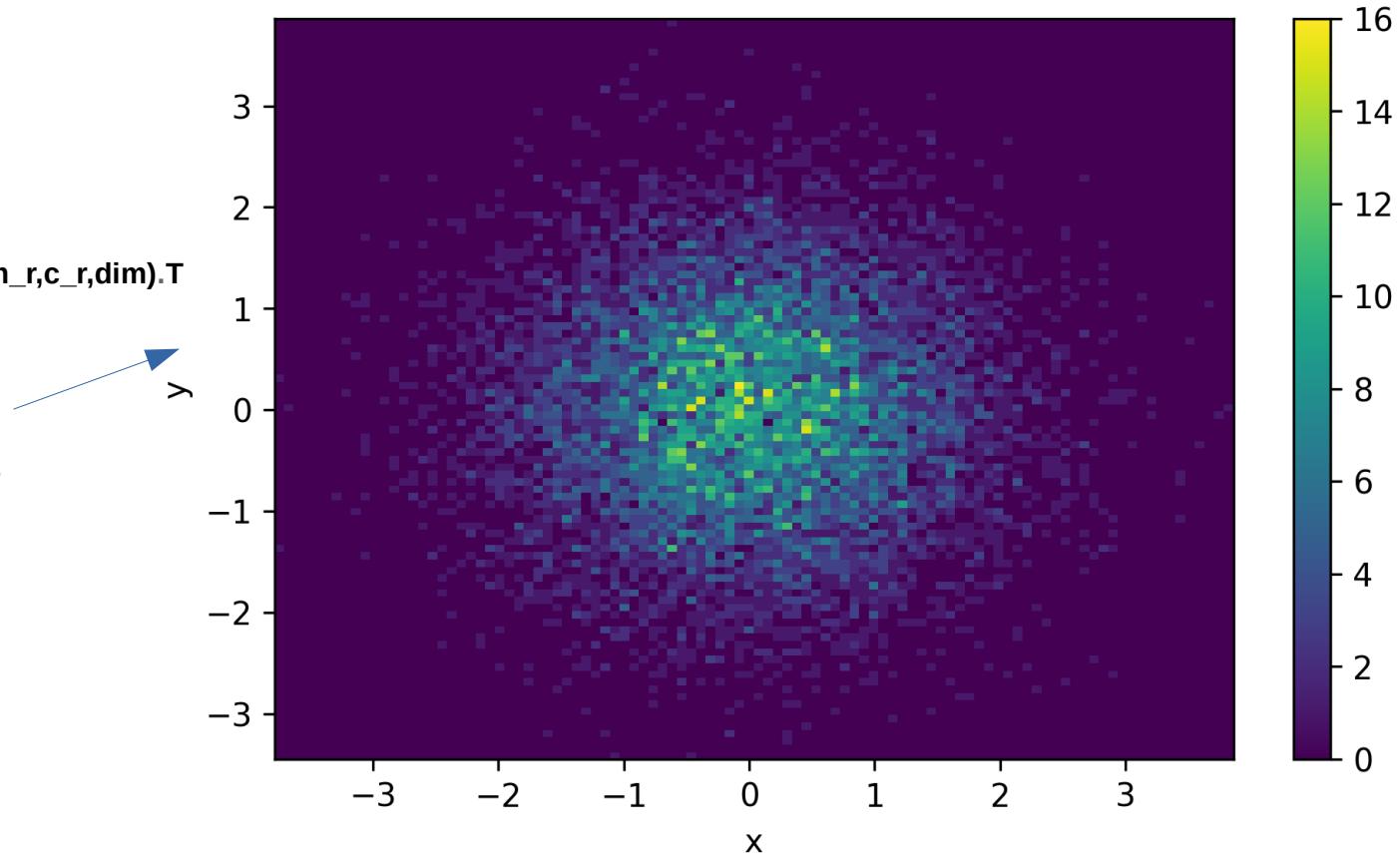
dim = 10000 #numero de pontos
m_r = [0,0] #media
c_r = [[1,0],[0,1]]

x,y = np.random.multivariate_normal(m_r,c_r,dim).T

plt.hist2d(x,y,bins=int(np.sqrt(dim)))
plt.xlabel("x")
plt.ylabel("y")

plt.colorbar() #add essa barra de cores

plt.tight_layout()
plt.savefig("hist2d.png",dpi = 300)
```



- Agora, um plot em três dimensões:

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')

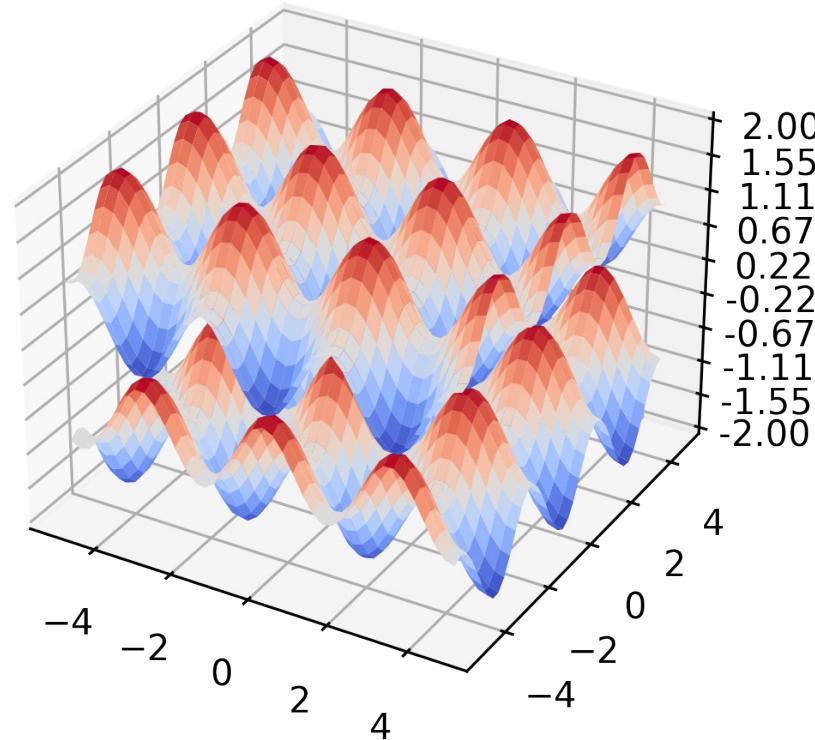
# gerando x,y e z
X = np.arange(-5, 5, 0.25) #definimos x
Y = np.arange(-5, 5, 0.25) #definir y
X, Y = np.meshgrid(X, Y) #criamos uma malha
R = np.sin(X)**2+np.cos(Y)**2 #forma do plot
Z = +R #parte superior
Z2 = -R #parte inferior

# Duas superfícies
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
surf = ax.plot_surface(X, Y, Z2, cmap=cm.coolwarm)

#Ajustes
ax.set_zlim(-npamax(Z),npamax(Z))
ax.xaxis.set_major_locator(LinearLocator(10))
ax.xaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.

plt.tight_layout()
plt.savefig("plot3d.png",dpi=300)
```



- Gráfico em 2d de contorno, tipo um histograma, mas fluido... Vejamos:

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

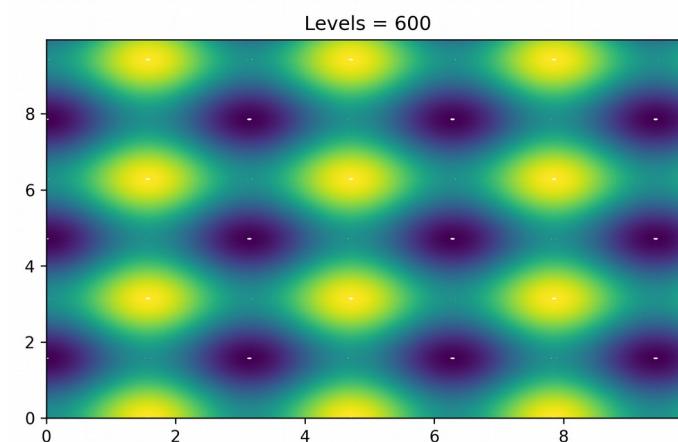
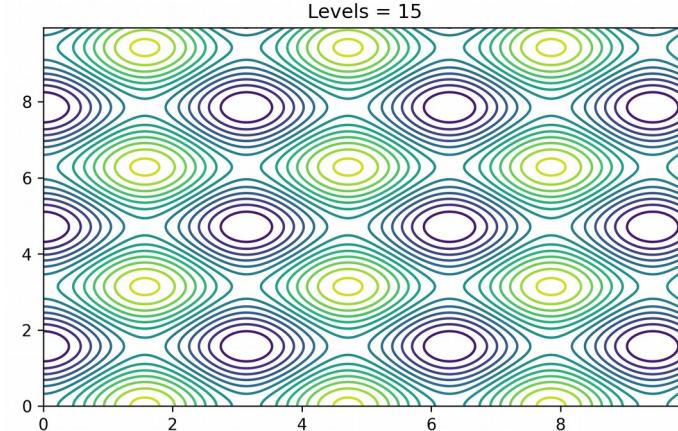
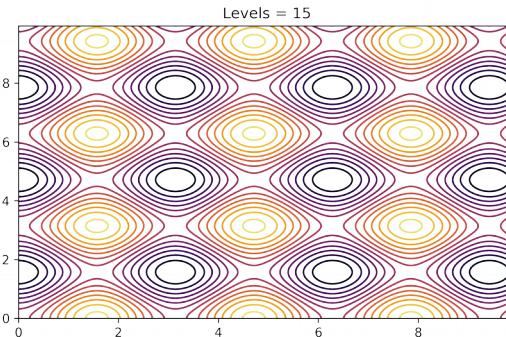
x = np.arange(0,10,0.05)
y = np.arange(0,10,0.05)

x,y = np.meshgrid(x,y)
z = np.sin(x)**2+np.cos(y)**2

fig = plt.contour(x,y,z,levels=600)
plt.colorbar()

plt.tight_layout()
plt.savefig("contour.png",transparent = True,dpi=300)
```

- Faça a figura abaixo:



- Por último, mas não menos importante o plot de vetores.

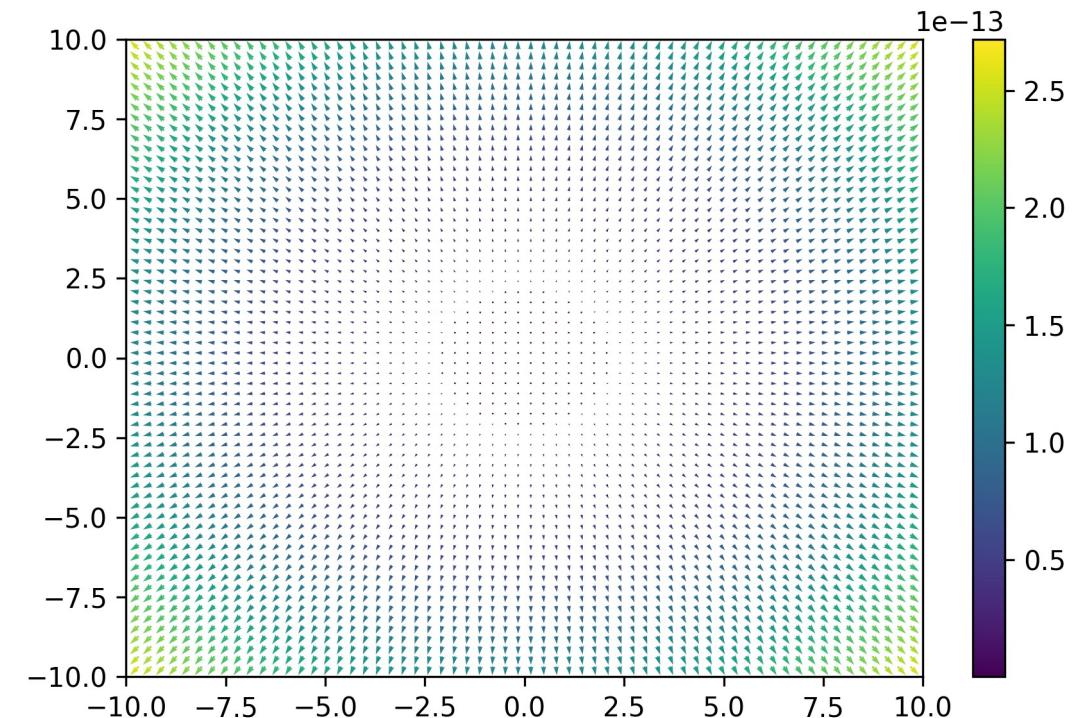
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

n_pts = 64
x = np.linspace(-10,10,n_pts)
y = np.linspace(-10,10,n_pts)

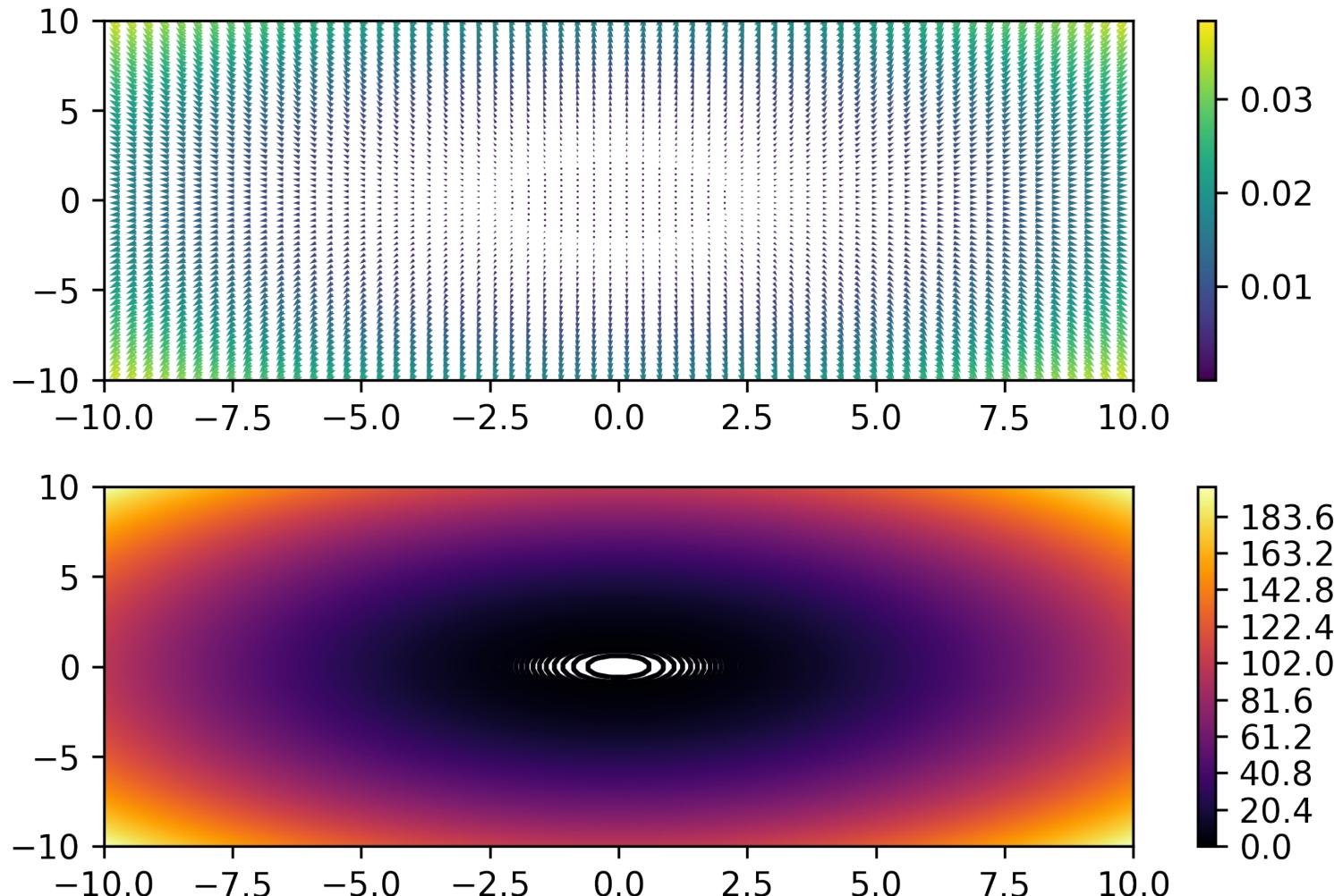
x,y = np.meshgrid(x,y)

ex = x/np.linalg.norm(x**2+y**2)**2
ey = y/np.linalg.norm(x**2+y**2)**2

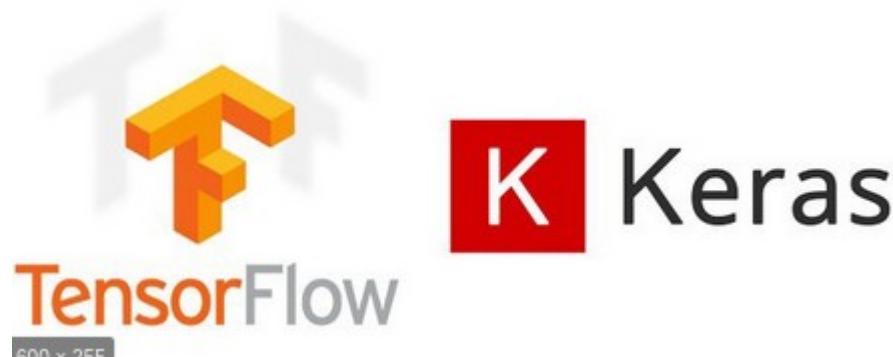
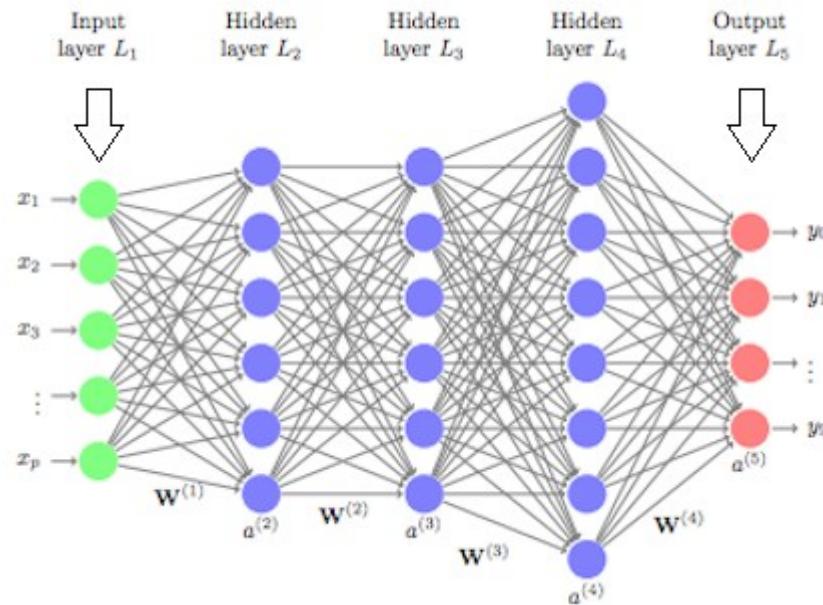
mod = ex**2+ey**2
VEC_PLOT = plt.quiver(x,y,ex,ey,mod)
plt.xlim(np.amin(x),np.amax(x))
plt.ylim(np.amin(y),np.amax(y))
plt.colorbar()
plt.tight_layout()
plt.savefig("vetores.png",dpi=300)
```



- Faça agora a figura abaixo (ou algo similar):

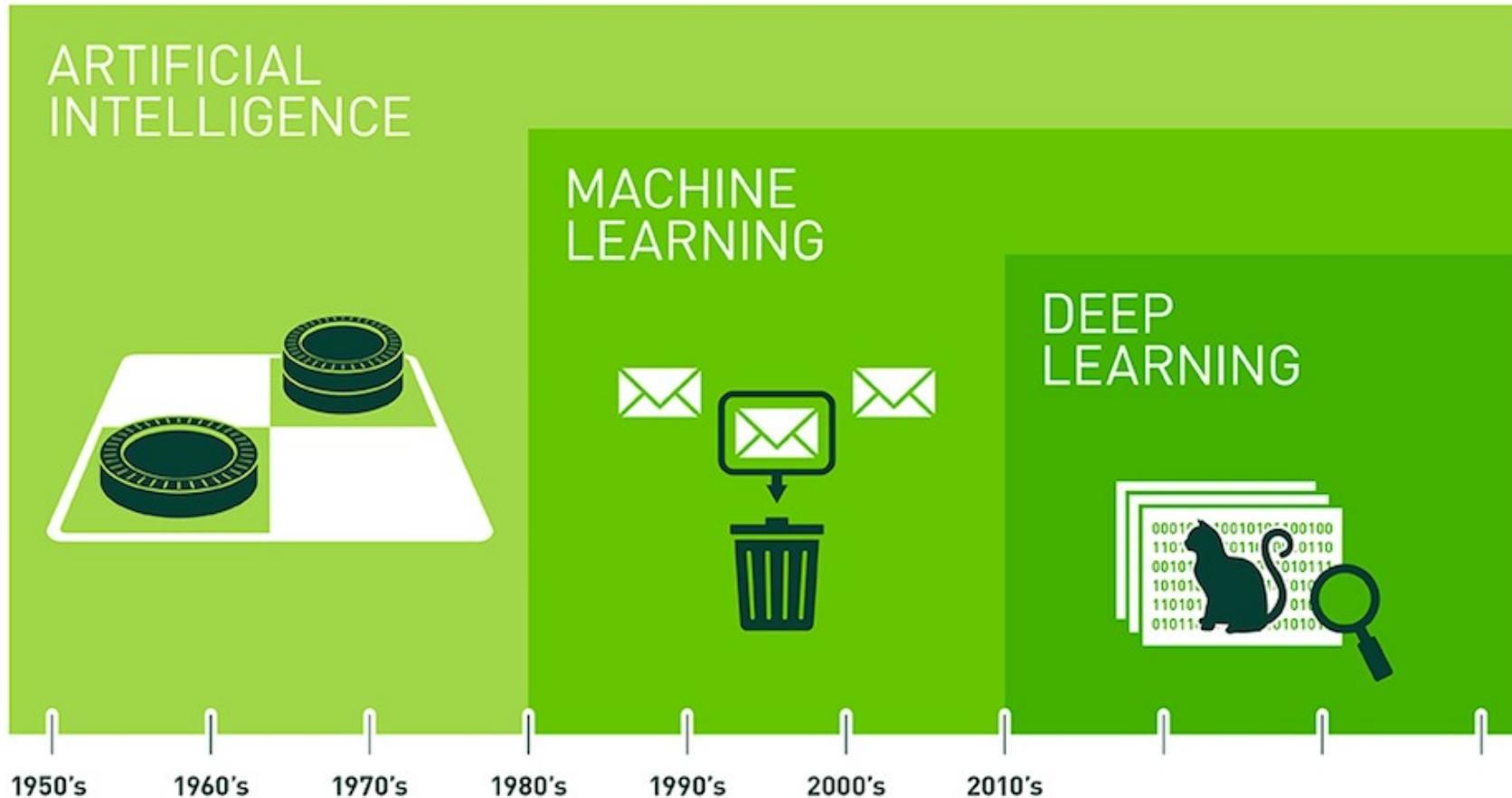


- Do que Estamos falando?
- Por que estudar?
- Redes Neurais
  - Supervisionadas e Não Supervisionadas
- Perceptrons e Sigmoides
- Aprendendo com o Gradiente
- Uma olhada no Keras
- Machine Learning na Prática
- Google Colab ([link](#))



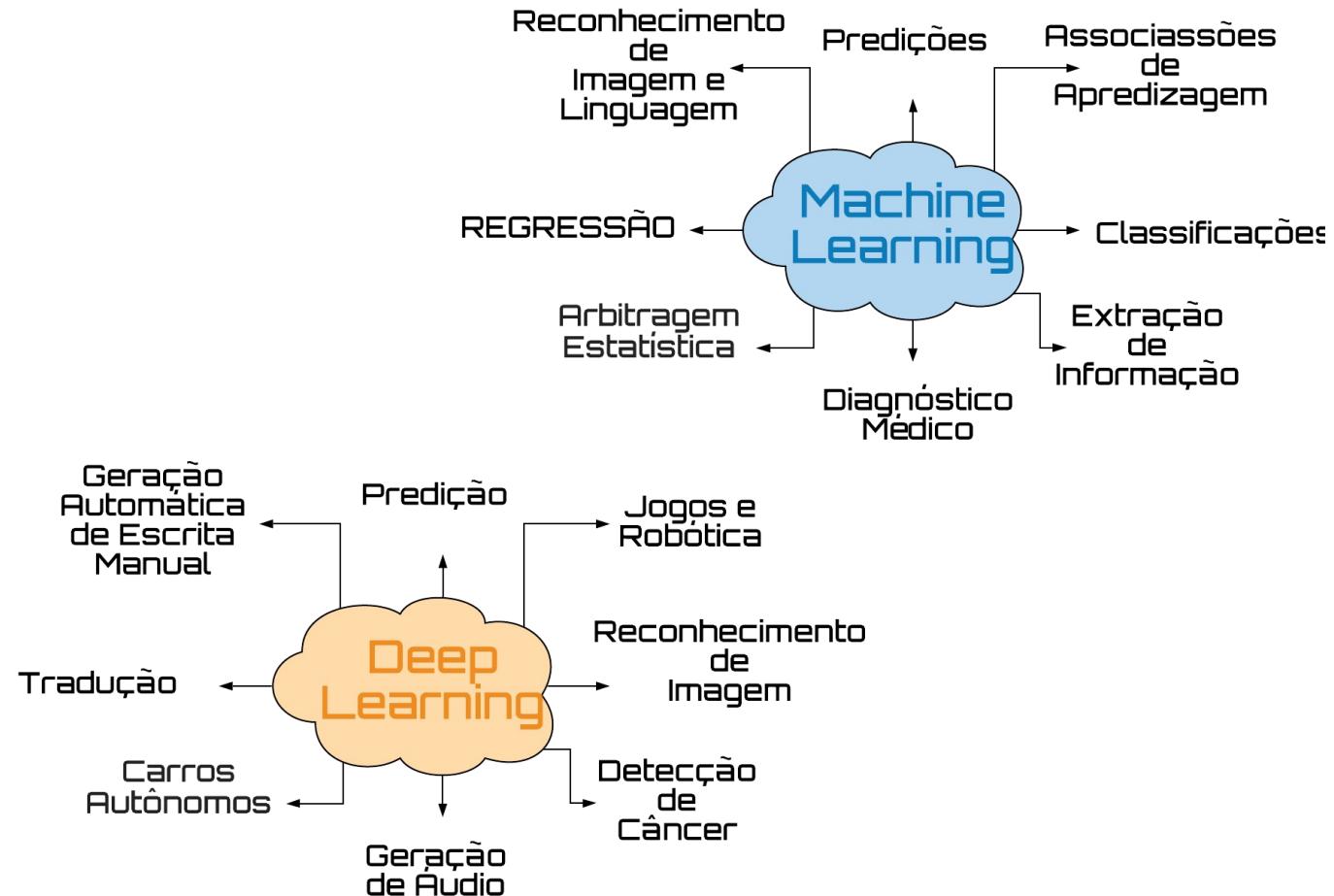
# IA, Machine Learning ou Deep Learning?

- Bem, para entendermos o restante da apresentação as ideias iniciais precisam ficar bem claras então, qual a diferença entre as redes neurais?



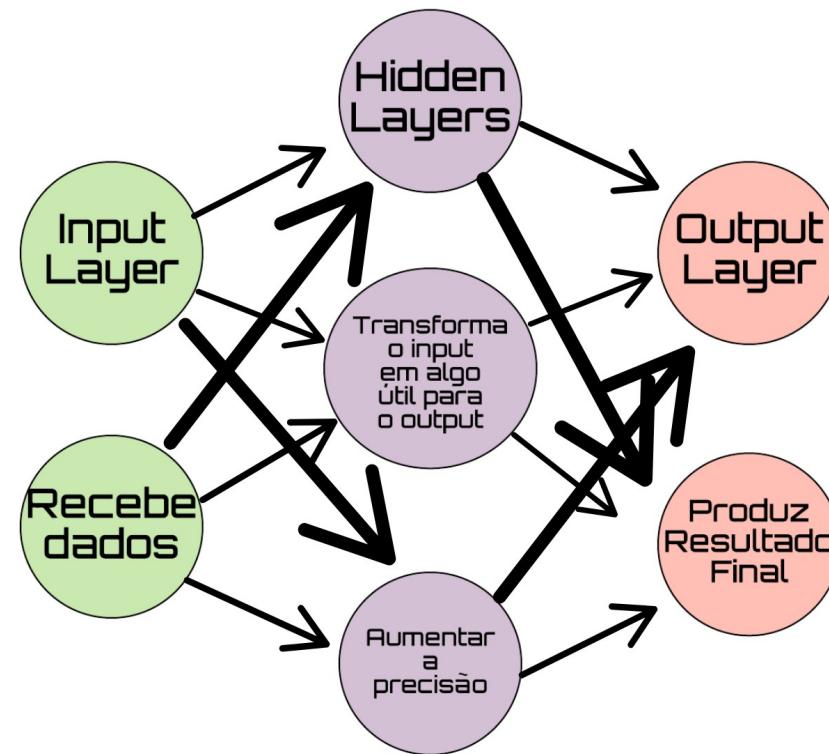
- **Inteligência Artificial**: tentativa de reproduzir o raciocínio humano por meio de máquinas/computadores (C3PO, jogos e etc);
- **Machine Learning**: alternativa à inteligência artificial, consiste no exercício de algoritmos bem estabelecidos para coletar, ler e aprender com os dados. Tratam-se de algoritmos robustos, capazes de encontrar inúmeras relações entre funções de inúmeras variáveis (regressão de múltiplas variáveis, relação de grandezas e etc);
- **Deep Learning**: Implementação mais robusta do Machine Learning com mais camadas de processamento (mais neurônios), geralmente aplicada para bancos de dados extensos (previsão do tempo, previsão do aluguel de casas, previsão do valor de ações e etc);

- Eae, o que fazemos com isso tudo?



- **E o que são redes neurais?**

- São uma série de algoritmos que imitam as operações de um cérebro humano para encontrar relações entre grandes quantidades de dados. Os neurônios são conectados em padrões complexos para processar os dados, um “neurônio” em uma rede neural nada mais é que uma função que coleta e classifica informações de acordo com uma arquitetura específica. Nada mais é que uma regressão linear parruda e muito elaborada de várias variáveis.

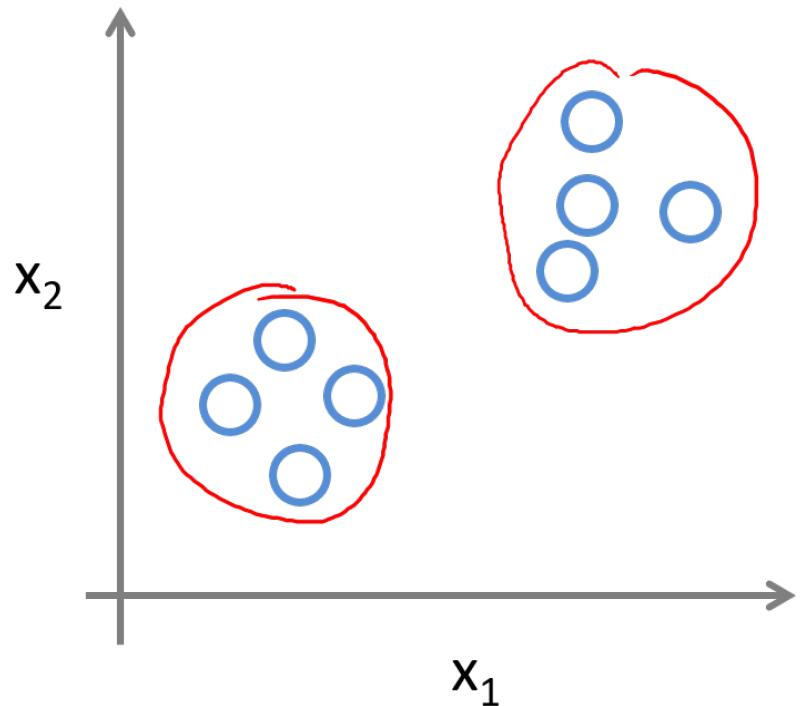
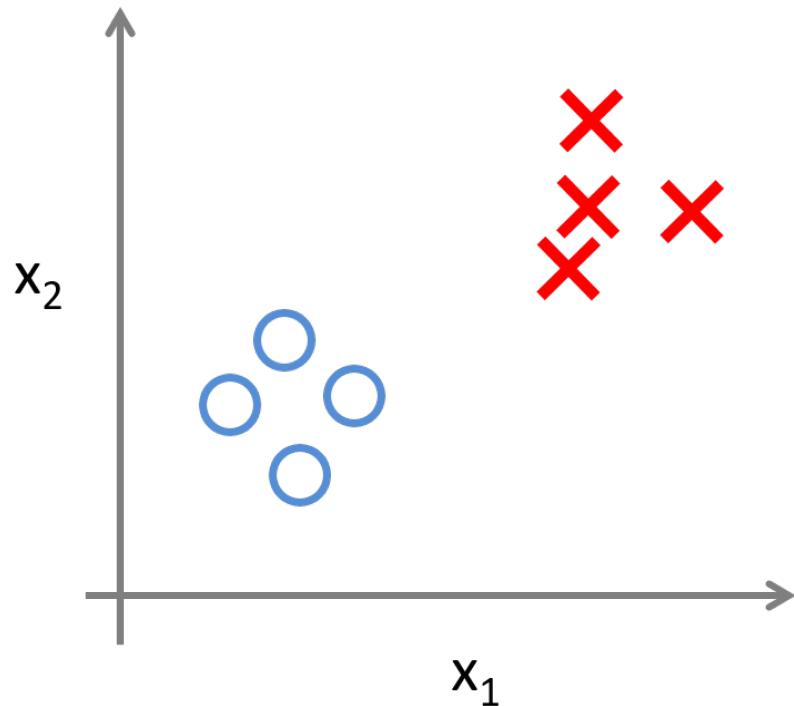


- **Tipos de Redes Neurais:**

- 1) **Aprendizado Supervisionado:** existe um conjunto de dados pré-rotulado pelo qual o computador passa e o algoritmo se modifica até que as previsões estejam próximas do esperado (regressão linear);
- 2) **Aprendizado Reforçado:** nesse tipo de algoritmo a rede neural é incentivada para resultados positivos e punida para resultados negativos (recomendação de vídeos no YouTube),
- 3) **Aprendizado Não Supervisionado:** Essa estratégia é utilizada quando não há um conjunto de dados previamente rotulado, a rede neural analisa o conjunto e em seguida classifica cada conjunto de dados em diferentes grupos (classificação de fotos).

## Supervised Learning

## Unsupervised Learning

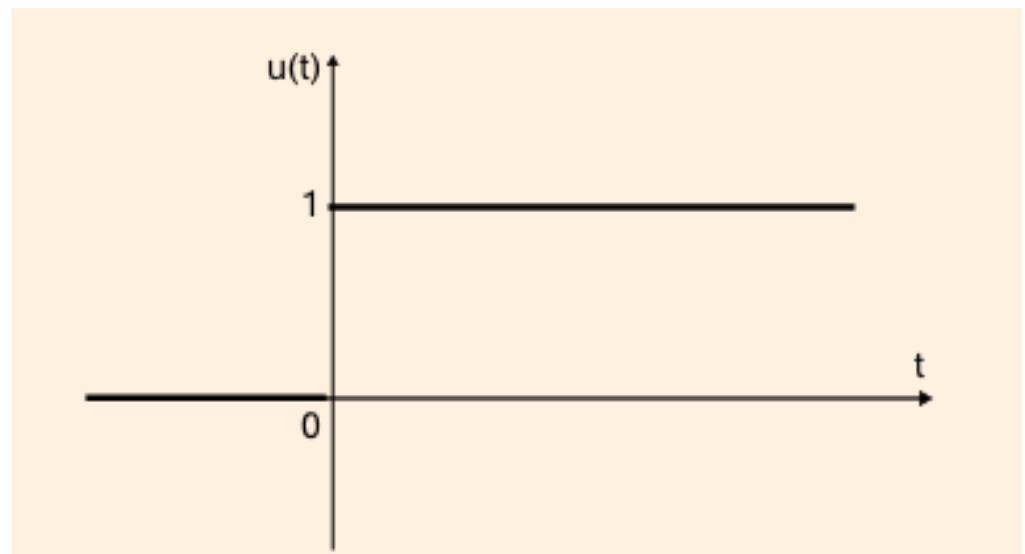


- **Os primeiros neurônios**
- Para iniciarmos será introduzido um tipo de neurônio artificial chamado perceptron;
- Como ele funciona? Trata-se de uma estrutura binária, ou seja, se um conjunto de condições for respeitado ele retorna “1”, caso contrário retorna “0”.

$$\theta = 0 \text{ se } \sum_j w_j x_j + b \leq 0$$

$$\theta = 1 \text{ se } \sum_j w_j x_j + b > 0$$

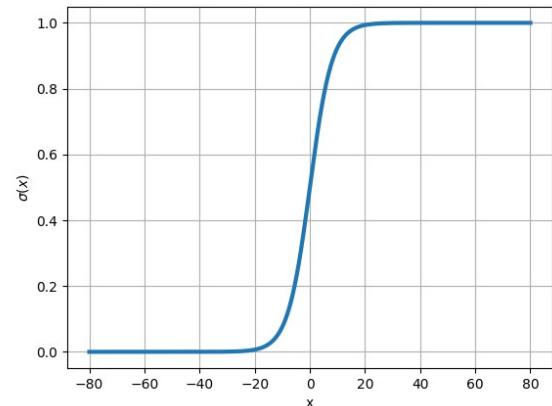
• onde “w” são os pesos, “b” o bias e “x” os dados de entrada



- Os perceptrons são fáceis de abstrair e tudo mais, mas são úteis? Não, definitivamente não são úteis. Eles possuem um problema fundamental do ponto de vista matemático, não trata-se de uma função diferenciável em primeira ordem, característica essa de extrema importância para os algoritmos utilizados.
- Uma boa alternativa entretanto são os sigmoids, visto que a função sigmoidal comporta-se como a função degrau para grandes valores e, próximo a zero possui curva suave.

$$f(x) = \frac{1}{1 + e^{w_j x_j - b_j}} \longrightarrow \Delta_{saída} \approx \sum_j \frac{\partial_{saída}}{\partial w_j} \Delta w_j + \frac{\partial_{saída}}{\partial b} \Delta b$$

- Assim, pequenas variações nos pesos causarão pequenas variações na saída.
- A relação acima implica que podemos variar lentamente os pesos para obter melhores resultados.



- Agora que já sabemos como os neurônios funcionarão, basta entender todo o resto. Bem, em ML queremos que a previsão da nossa máquina seja próxima ao resultado real, para avaliar essa proximidade, definiremos uma coisa chamada **função custo**.
- Uma **função custo** comum é o desvio quadrático:  $C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a||^2$ 
  - $y(x)$  é o valor conhecido;
  - $a(w,b)$  é o valor previsto
  - $n$  o número de amostras
  - o somatório percorre todos os valores de “ $x$ ”.
- Bem, aumentar a precisão implica em diminuir a função custo, temos então a maneira como nosso programa aprenderá, variar “ $w$ ” e “ $b$ ” de modo que minimize a **função custo**. Faremos isso através do gradiente da função, vamos lá...

- Bem, embora normalmente trabalharemos com milhões de dados, imaginemos aqui por simplicidade o vetor:  $\eta = (w, b)$

$$\Delta C = \nabla C \Delta \eta$$

- Reescrevendo  $\Delta \eta$ :

$$\Delta \eta = -\xi \nabla C, \xi \approx 0$$

- O que nos leva a:  $w = w - \xi \nabla C$  e  $b = b - \xi \nabla C$

- O esquema acima evidentemente nos levará aos valores que minimizam a função custo, ou seja, esse algoritmo nos levará ao mínimo global de uma função com infinitas variáveis!
- O fator  $\xi$  é conhecido como parâmetro de aprendizado, se muito grande o programa dará saltos em seu aprendizado, caso seja muito pequeno o programa não aprenderá nunca (lembre-se sempre de pensar no custo computacional dessas coisas).

- Só precisamos então encontrar o gradiente. Mas como? Há um algoritmo fundado sobre cinco equações que nos possibilita encontrá-lo chamado “**Back Propagation**”:

1º  $\Rightarrow$  Escolha os dados para o primeiro  $a^L$

2º  $\Rightarrow$  Para cada  $L$  encontre  $z^L = w^L a^{L-1} + b^L$ ,  $a^L = \sigma(z^L)$

3º  $\Rightarrow$  Calcule os erros com  $\delta^L = \nabla_a C \cdot \sigma(z^L)$

4º  $\Rightarrow$  Propague os erros para tras com  $\delta^L = (w^{L+1} \delta^{L+1}) \sigma(z^L)$

5º  $\Rightarrow$  O gradiente da função custo :  $\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1} \delta_j^L$  e  $\frac{\partial C}{\partial b_j^L} = \delta_j^L$

# Keras e TensorFlow

- Bem, o Keras é um pacote que roda com o backend do tensorflow.
- Extremamente otimizado e, paraleliza processos entre CPU e GPU nativamente

`from keras import models` → importa modelos possíveis para sua rede, podem ser modelos próprios ou o sequencial

`from keras import loss` → importa funções custo, como quadrática absoluta, quadrática média, crossentropia binária

`from keras import layers` → importa um pacote capaz de construir camadas

`from keras import metrics` → importa métricas para avaliação da rede neural

`from keras import optimizers` → importa funções de otimização como gradiente estocástico descendente e RMSprop

- Façamos uma rede juntos, ao vivo!

```
import numpy as np
from keras import layers, models, losses, optimizers
```

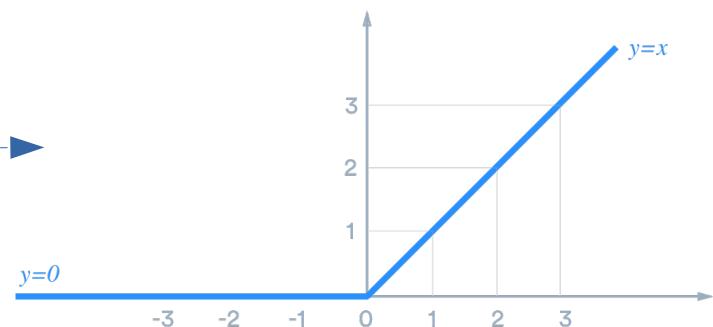
```
n = 10**4
x = np.linspace(0,100,n)
a = 5
b = -5
```

```
x_val = np.random.choice(x,int(0.4*n))
x_tre = np.setdiff1d(x,x_val)
```

```
y_val = a*x_val +b+ np.random.random(len(x_val))*0.05 +np.sin(x_val)
y_val = a*x_tre +b+ np.random.random(len(x_tre))*0.05 +np.sin(x_tre)
```

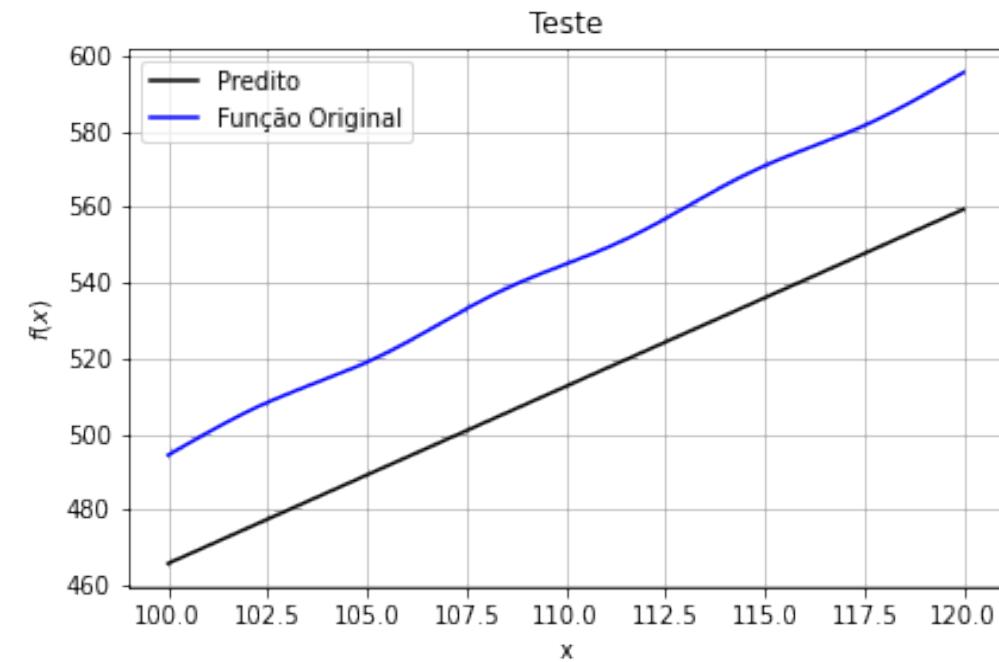
```
rede = models.Sequential()
rede.add(layers.Dense(32,activation='relu',input_shape=[1]))
rede.add(layers.Dense(32,activation='relu'))
rede.add(layers.Dense(1))
rede.compile(optimizer = optimizers.RMSprop(lr = 0.05), loss = losses.mean_squared_error)

hist = rede.fit(x_tre,y_tre,epochs=180,batch_size=25,validation_data=(x_val,y_val))
```

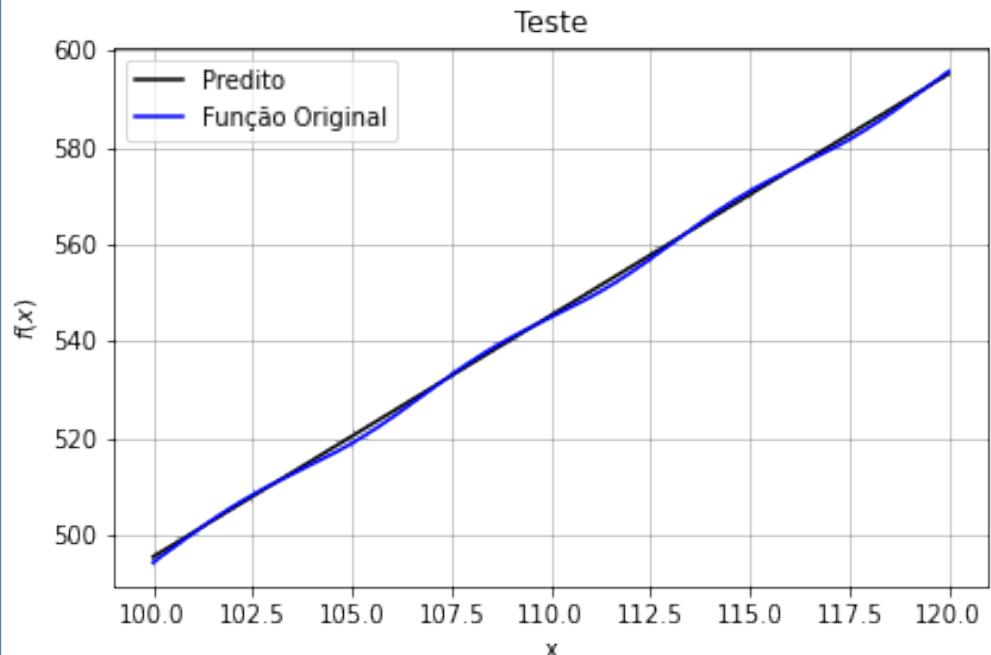


# A importância dos parâmetros corretos:

epochs = 180  
batch\_size = 25  
lr = 0.05



epochs = 75  
batch\_size=100  
lr = 0.001



# E agora ?

- Aos que se interessaram pelo potencial numérico do Python : Fundamentals of Engineering Numerical Analysis (P. Moin), High Performance Python (Gorelick & Ozsvald), <https://compphysics.readthedocs.io/> (Gerson J. Ferreira), NUMERICAL METHODS WITH FORTRAN IV CASE STUDIES (Dorn McCracken), Applied Numerical Linear Algebra (James W. Demmel) e Problemas Clássicos de Ciências da Computação com Python (David Kopec).
- Para Machine Learning : Deep Learning With Python (Fancois Chollet), Neural Networks and Deep Learning (Michael Nielsen) e An Introduction to Statistical Learning (Gareth James).
- Plots bonitos, avançados e interativos : Plotly ([plotly](#)) e Seaborn ([seaborn](#)).
- Achou legal programar, mas não curtiu o Python ? Algumas dicas : Matlab com Aplicações em Engenharia (Amos Gilat), Performance Optimization (Stefan Goedecker), High Perfomance Computing (Kevin Dowd).