Python
Aula 12 - Matplotlib e Scipy

Prof. Eduardo Campos (CEFET-MG)

Slides do prof. João Silva da UFRJ

Matplotlib - Básico

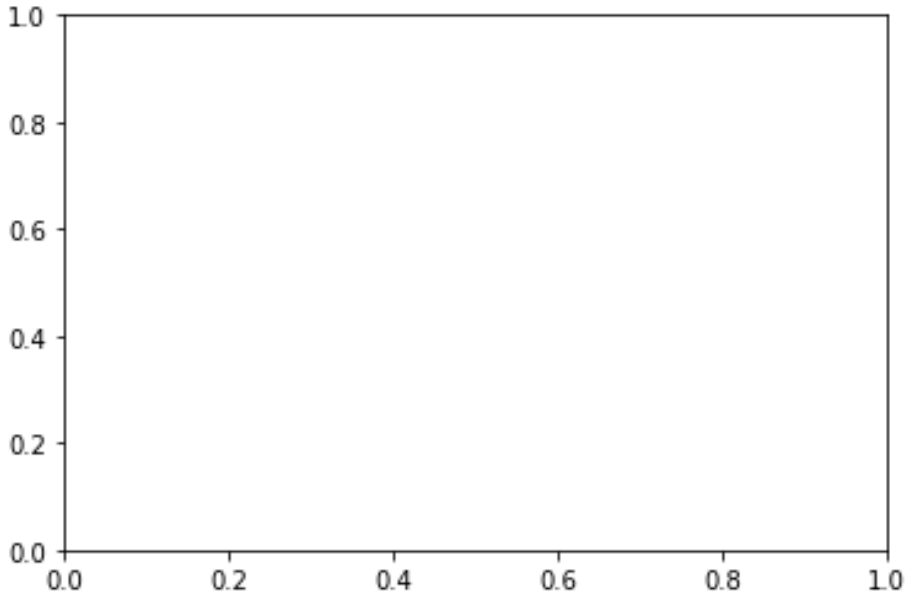
Queremos plotar a função *seno* para 200 pontos no intervalo $[0, 10)$

Matplotlib - Básico

Queremos plotar a função *seno* para 200 pontos no intervalo $[0, 10)$

```
1 import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
2
3 fig, ax = plt.subplots()
```

- **fig, ax = plt.subplots()**: retorna um par de elemento, onde:
  - **fig** é uma instância de figura ("quadro em branco")
  - **ax** é um frame no qual você pode plotar algo



Matplotlib - Básico

Queremos plotar a função *seno* para 200 pontos no intervalo $[0, 10)$

```
1 import numpy as np          # biblioteca Numpy
2 import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4 fig, ax = plt.subplots()
5 x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6 y = np.sin(x)
```

- Criamos os arrays **x** e **y** usando o np.linspace.

Matplotlib - Básico

Queremos plotar a função *seno* para 200 pontos no intervalo $[0, 10)$

```
1 import numpy as np        # biblioteca Numpy
2 import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4 fig, ax = plt.subplots()
5 x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6 y = np.sin(x)
```

- Criamos os arrays **x** e **y** usando o np.linspace.

Agora, queremos plotar o gráfico:

```
1 import numpy as np        # biblioteca Numpy
2 import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4 fig, ax = plt.subplots()
5 x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6 y = np.sin(x)
7
8 ax.plot(x, y, 'b-', linewidth=5) # plotando o grafico
```

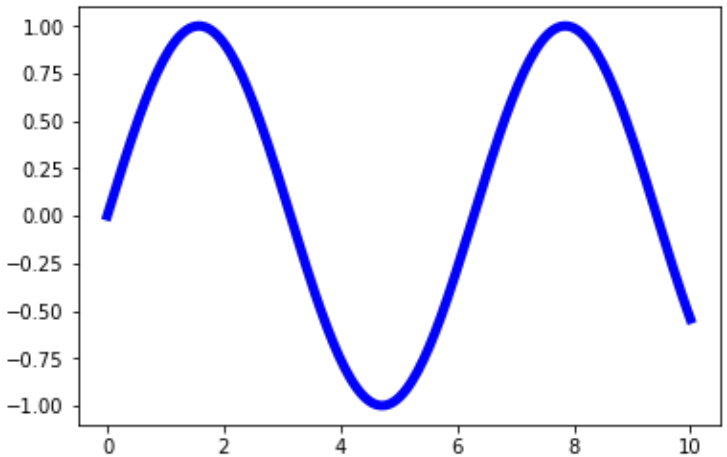
- x e y são as coordenadas dos pontos
- $'b-'$ define a cor azul (b) e o estilo da linha (-)
- *linewidth* define a largura da linha

Matplotlib - Básico

Queremos plotar a função *seno* para 200 pontos no intervalo $[0, 10)$

```
1 import numpy as np          # biblioteca Numpy
2 import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4 fig, ax = plt.subplots()
5 x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6 y = np.sin(x)
7
8 ax.plot(x, y, 'b-', linewidth=5) # plotando o grafico
```

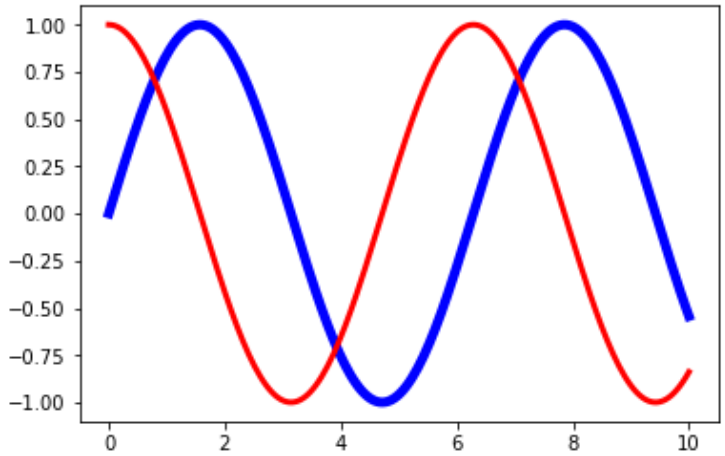
- x e y são as coordenadas dos pontos
- $'b-'$ define a cor azul (b) e o estilo da linha (-)
- *linewidth* define a largura da linha



Matplotlib - Básico

Podemos plotar mais de uma função em um mesmo gráfico:

```
1  import numpy as np          # biblioteca Numpy
2  import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6  y = np.sin(x)
7  z = np.cos(x)
8
9  ax.plot(x, y, 'b-', linewidth=5) # plotando o seno
10 ax.plot(x, z, 'r-', linewidth=3) # plotando o coseno
```



Matplotlib - Básico

Formatos - Linhas e Pontos

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Matplotlib - Básico

Formatos - Cores

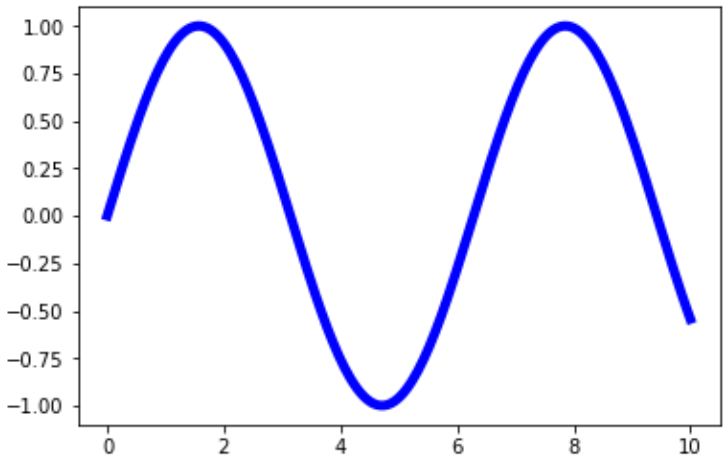
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Matplotlib - Básico

Podemos querer mostrar uma figura ou salvá-la em um arquivo:

- **show()**: mostra a figura plotada
- **savefig(...)**: salva a figura em um arquivo

```
1  import numpy as np          # biblioteca Numpy
2  import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6  y = np.sin(x)
7
8  ax.plot(x, y, 'b-', linewidth=5) # plotando o grafico
9
10 plt.show()          # mostrando o grafico
```

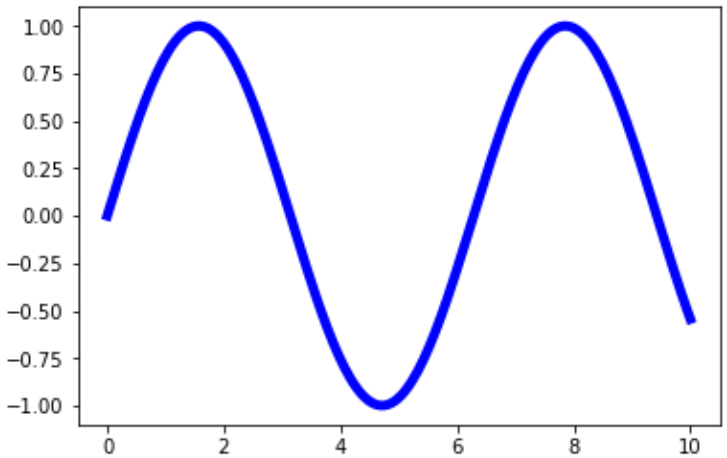


Matplotlib - Básico

Podemos querer mostrar uma figura ou salvá-la em um arquivo:

- **show()**: mostra a figura plotada
- **savefig(...)**: salva a figura em um arquivo

```
1  import numpy as np          # biblioteca Numpy
2  import matplotlib.pyplot as plt  # biblioteca matplotlib.pyplot
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0, 10, 200)      # 200 pontos do intervalo [0,10)
6  y = np.sin(x)
7
8  ax.plot(x, y, 'b-', linewidth=5) # plotando o grafico
9
10 plt.savefig('/home/joao/Desktop/figura.png')  # mostrando o grafico
```



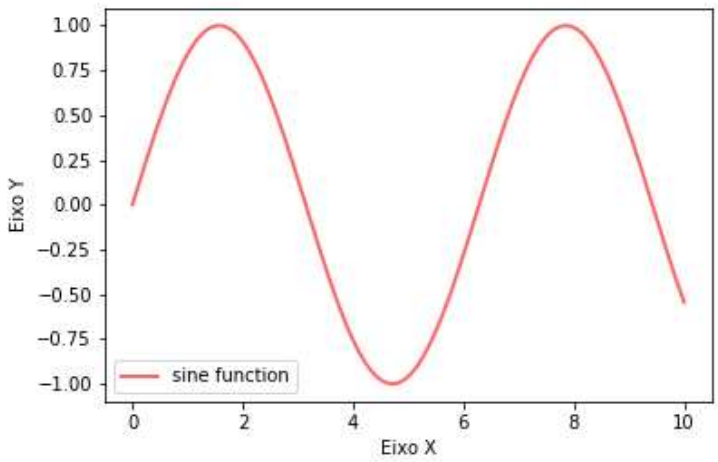
Matplotlib - Básico

Exercício:

- Teste outras cores e formatos para o exemplo da função *seno*. Altere também o número de pontos.
- Plote as funções *seno* e *coseno* no mesmo gráfico, em cores e formatos diferentes.
- Plote as funções trigonométricas *coseno* (cos) e *tangente* (tan) de forma que as cores e formatos de cada plot sejam distintas. Varie também o número de pontos usados na geração dos gráficos. Salve seus gráficos em arquivo.

Matplotlib - Básico - Rótulos e Legendas

```
1  fig, ax = plt.subplots(figsize=(10, 5)) # podemos mudar o tamanho do grafico gerado
2  x = np.linspace(0, 10, 200)
3  y = np.sin(x)
4  ax.plot(x,y,'r-',linewidth=2,label='sine function', alpha=0.6) # label da figura
5
6  plt.xlabel('Eixo X')    # rotulo do eixo X
7  plt.ylabel('Eixo Y')    # rotulo do eixo Y
8
9  ax.legend()     # inclui a legenda
10 plt.show()
```

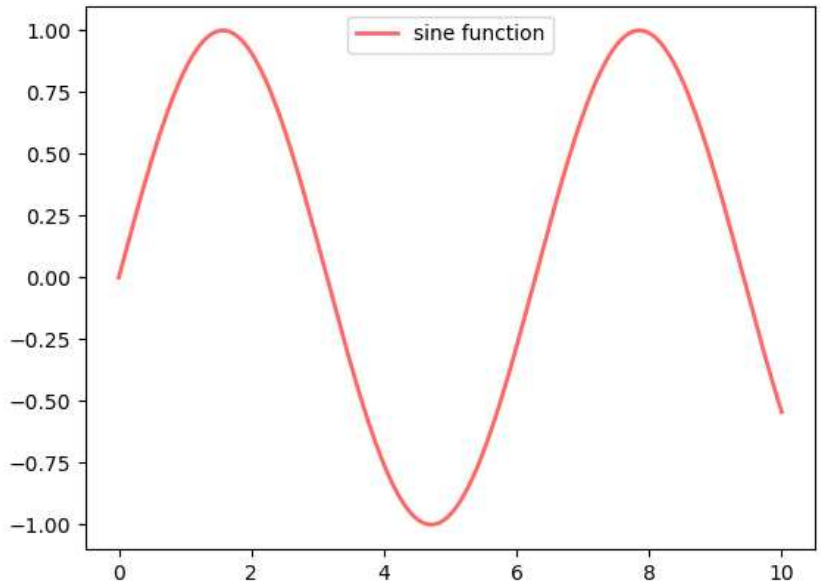


- **figsize**: permite definir o tamanho da figura
- **label**: acrescenta um rótulo para a função
- **plt.xlabel** e **plt.ylabel**: rótulo dos eixos x e y
- **alpha**: grau de "transparência" da linha (quanto menor, mais transparente)
- **legend**: inclui a legenda na figura

Matplotlib - Básico - Rótulos e Legendas

```
1 fig, ax = plt.subplots(figsize=(10, 5))
2
3 x = np.linspace(0, 10, 200)
4 y = np.sin(x)
5
6 ax.plot(x, y, 'r-', linewidth=2, label='sine function', alpha=0.6)
7
8 ax.legend(loc='upper center')  # mudamos aqui
9 plt.show()
```

Podemos mudar a posição da legenda na figura:



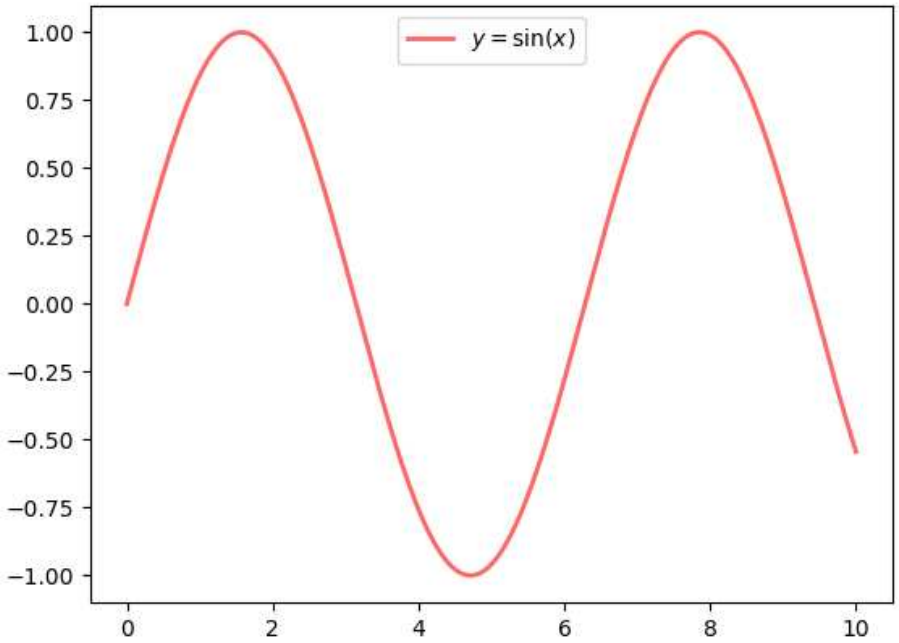
Matplotlib - Básico - Rótulos e Legendas

```
1 fig, ax = plt.subplots(figsize=(10, 5))
2
3 x = np.linspace(0, 10, 200)
4 y = np.sin(x)
5
6 ax.plot(x, y, 'r-', linewidth=2, label='sine function', alpha=0.6)
7
8 ax.legend(loc='upper center')  # mudamos aqui
9 plt.show()
```

String	Código
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower,right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper,center'	9
'center'	10

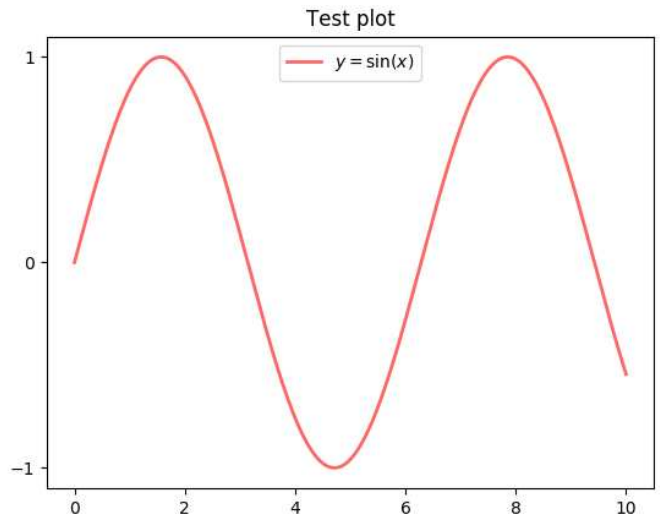
Matplotlib - Básico - Rótulos e Legendas

```
1  fig , ax = plt.subplots(figsize=(10, 5))
2
3  x = np.linspace(0, 10, 200)
4  y = np.sin(x)
5
6  ax.plot(x,y,'r-',linewidth=2,label=r'$y=\sin(x)$',alpha=0.6)  # mudamos aqui
7
8  ax.legend(loc='upper center')
9
10 plt.show()
```



Matplotlib - Básico - Rótulos e Legendas

```
1  fig , ax = plt.subplots()
2  x = np.linspace(0, 10, 200)
3  y = np.sin(x)
4
5  ax.plot(x, y, 'r-', linewidth=2, label=r'$y=\sin(x)$', alpha=0.6)
6  ax.legend(loc='upper center')
7
8  ax.set_yticks([-1, 0, 1])     # acrescentamos esta linha
9  ax.set_title('Test plot')     # acrescentamos esta linha
10
11 plt.show()
```



- **ax.set_yticks()**: elementos que formam o eixo y
- **ax.set_title()**: título da figura

Matplotlib - Básico

Exercícios:

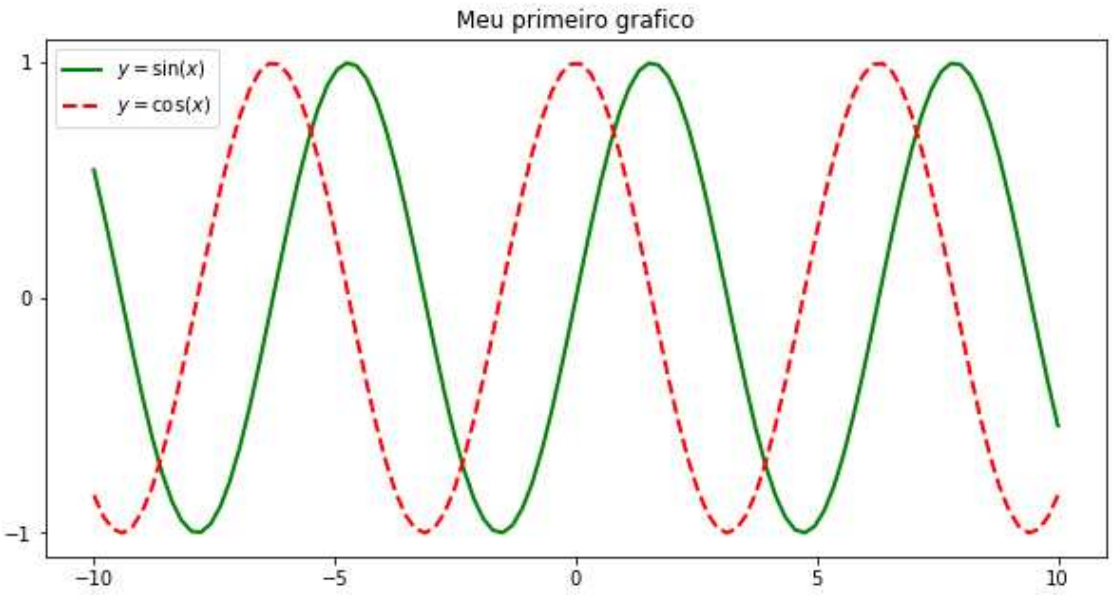
- Crie um array com números do intervalo $[0., 5.)$ (step $= 0.2$), e plote as funções
  - $f(x) = x$ (formato: traços vermelhos)
  - $g(x) = x^2$ (quadrados azuis)
  - $h(x) = x^3$ (triângulos verdes).

  Fazer duas versões, uma com eles separados e outra em um só gráfico. Inclua a legenda para as funções e salve as imagens em arquivos.

Matplotlib - Básico

Exercício

- Considerando que você está usando 100 pontos no intervalo $[-10, 10]$, plote as funções *seno* e *coseno* para gerar o seguinte gráfico:

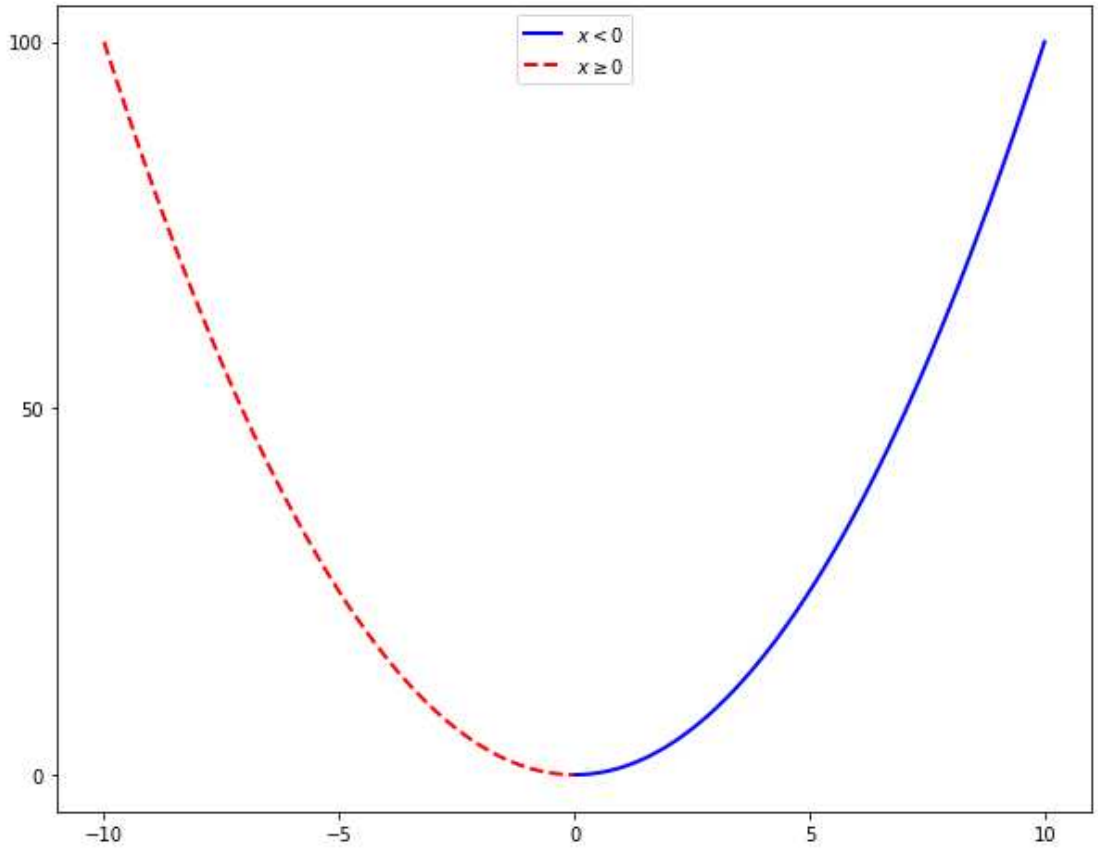


O tamanho da figura é 10 x 5. Salve seu gráfico em arquivo.

Matplotlib - Básico

Exercício

- Plote o gráfico abaixo. Use 100 pontos e o tamanho da figura de 10 x 8. Salve seu gráfico em arquivo.



Matplotlib - Funções Descontínuas

Considere a seguinte função:

$$y = \begin{cases} x & \text{if } x < 0.5 \\ 1 + x & \text{if } x \geq 0.5 \end{cases}$$

Matplotlib - Funções Descontínuas

Considere a seguinte função:

$$y = \begin{cases} x & \text{if } x < 0.5 \\ 1 + x & \text{if } x \geq 0.5 \end{cases}$$

Podemos plotar a função usando o seguinte código:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  fig, ax = plt.subplots()
4  x=np.linspace(0,1,100)     # array x
5  y=np.zeros(100)      # array y, inicializado com zero
6
7  for i in range(100):     # definindo a funcao
8      if x[i]<0.5:
9          y[i] = x[i]
10     else:
11         y[i]=1+x[i]
12 ax.plot(x, y, '-o')
13 plt.savefig('/home/joao/Desktop/descont1.png')
```

Matplotlib - Funções Descontínuas

Considere a seguinte função:

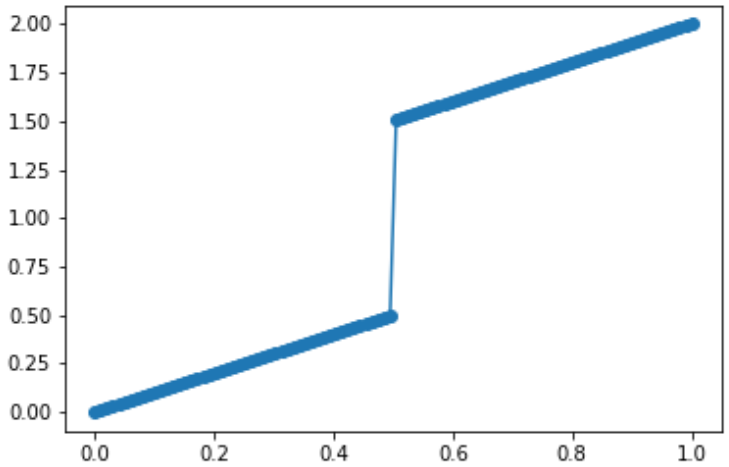
$$y = \begin{cases} x & \text{if } x < 0.5 \\ 1 + x & \text{if } x \geq 0.5 \end{cases}$$

Podemos simplificar a geração da função usando o fato que estamos trabalhando com arrays:

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  fig, ax = plt.subplots()
5  x=np.linspace(0,1,100)     # array x
6  y=np.zeros(100)      # array y, inicializado com zero
7
8  y[x<0.5] = x[x<0.5]    # forma alternativa para definir a funcao
9  y[x>=0.5] = 1 + x[x>=0.5]
10
11 ax.plot(x, y, '-o')
12 plt.savefig('/home/joao/Desktop/descont2.png')
```

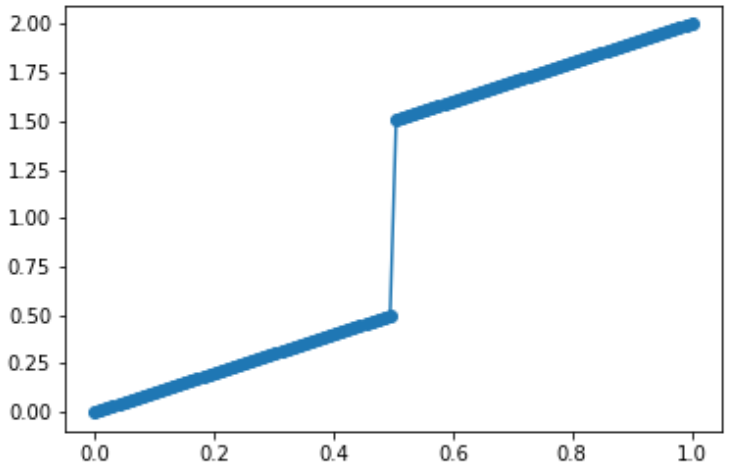
Matplotlib - Funções Descontínuas

Nos dois programas obtemos o mesmo gráfico:



Matplotlib - Funções Descontínuas

Nos dois programas obtemos o mesmo gráfico:



- Por que isto ocorreu?
- Como resolver este problema?

Matplotlib - Funções Descontínuas

- Vamos usar *np.nan* para quebrar a linha em múltiplos seguimentos

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0,1,100)
6  y = np.zeros(100)
7
8  y[x < 0.5] = x[x < 0.5]
9  y[x >= 0.5] = 1 + x[x >= 0.5]
10
11 pos = np.where(np.abs(np.diff(y)) >= 0.5)[0]   # linha nova
12
13 x[pos] = np.nan                 # linha nova
14 y[pos] = np.nan                 # linha nova
15
16 ax.plot(x, y, '-o')
17 plt.savefig('/home/joao/Desktop/descont3.png')
```

- **np.diff(y)** : diferença entre os elementos de um array (out[n] = y[n+1]-y[n])
- **np.abs** : retorna o valor absoluto do seu argumento
- **np.where(condicao[,x,y])** : retorna elementos (de *x* ou *y*) em um *ndarray* ou tupla de *ndarrays*, dependendo de *condicao*

Matplotlib - Funções Descontínuas

- Vamos usar *np.nan* para quebrar a linha em múltiplos seguimentos

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0,1,100)
6  y = np.zeros(100)
7
8  y[x < 0.5] = x[x < 0.5]
9  y[x >= 0.5] = 1 + x[x >= 0.5]
10
11 pos = np.where(np.abs(np.diff(y)) >= 0.5)[0]   # linha nova
12
13 x[pos] = np.nan                 # linha nova
14 y[pos] = np.nan                 # linha nova
15
16 ax.plot(x, y, '-o')
17 plt.savefig('/home/joao/Desktop/descont3.png')
```

- **x[pos] = np.nan** e **y[pos] = np.nan** : substitui o ponto onde ocorre a descontinuidade por *np.nan*

Matplotlib - Funções Descontínuas

- Vamos usar *np.nan* para quebrar a linha em múltiplos seguimentos

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  fig, ax = plt.subplots()
5  x = np.linspace(0,1,100)
6  y = np.zeros(100)
7
8  y[x < 0.5] = x[x < 0.5]
9  y[x >= 0.5] = 1 + x[x >= 0.5]
10
11 pos = np.where(np.abs(np.diff(y)) >= 0.5)[0]   # linha nova
12
13 x[pos] = np.nan                 # linha nova
14 y[pos] = np.nan                 # linha nova
15
16 ax.plot(x, y, '-o')
17 plt.savefig('/home/joao/Desktop/descont3.png')
```

- **x[pos] = np.nan** e **y[pos] = np.nan** : substitui o ponto onde ocorre a descontinuidade por *np.nan*

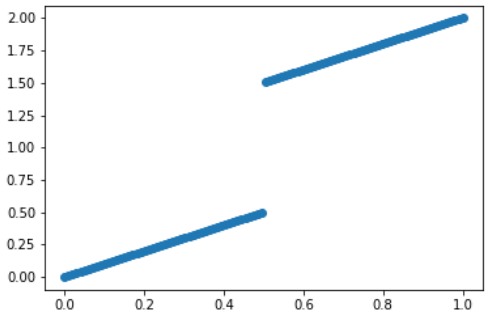
- Note que o ponto foi perdido!

Matplotlib - Funções Descontínuas

- Vamos usar *np.nan* para quebrar a linha em múltiplos seguimentos

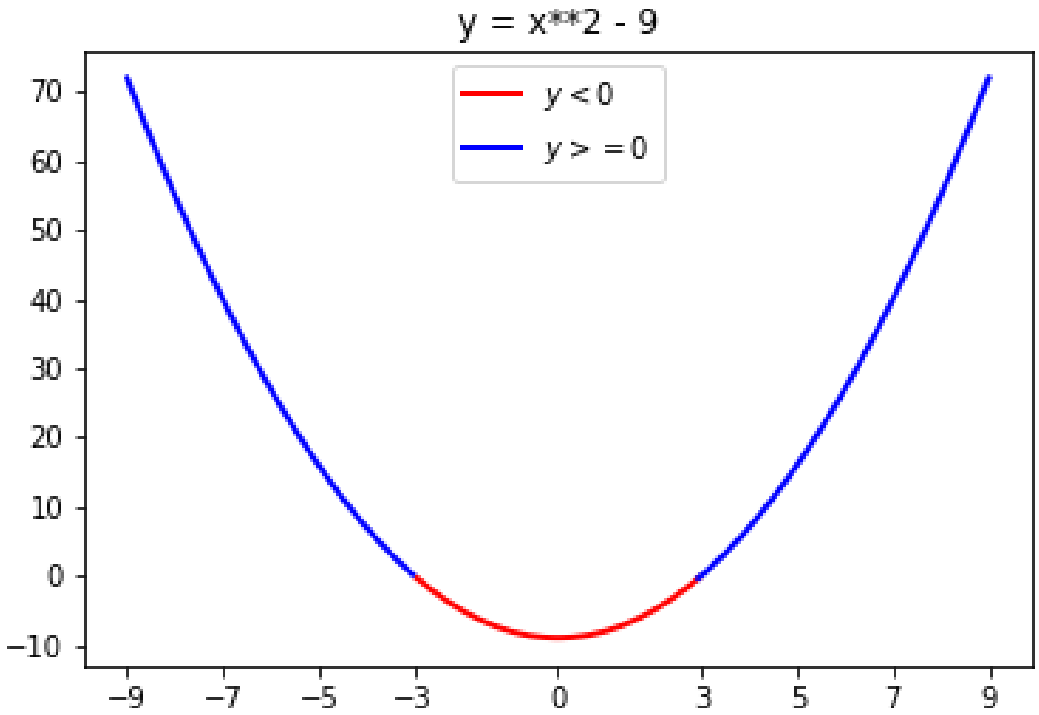
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  fig, ax = plt.subplots()
4  x=np.linspace(0,1,100)
5  y=np.zeros(100)
6  y[x<0.5] = x[x<0.5]
7  y[x>=0.5] = 1 + x[x>=0.5]
8
9  pos = np.where(np.abs(np.diff(y)) >= 0.5)[0]+1   # acrescentamos + 1 a posicao
10 x = np.insert(x, pos, np.nan)                    # acrescentamos ponto de quebra
11 y = np.insert(y, pos, np.nan)                    # acrescentamos ponto de quebra
12 ax.plot(x, y, '-o')
13 plt.savefig('/home/joao/Desktop/descont3.png')
```

- **x = np.insert(x, pos, np.nan)** e **y = np.insert(y, pos, np.nan)**: acrescentamos um novo ponto no array ao invés de substituí-lo.



Matplotlib - Funções Descontínuas

Exercício: Faça uma função que gere o seguinte gráfico (use 200 pontos e valores de $x \in [-10, 10]$):



Scipy

SciPy é uma biblioteca construída sobre NumPy, aumentando o poder de manipulação e visualização de dados, tornando este ambiente competitivo com outros sistemas como MATLAB, Octave e SciLab.
Ela fornece uma série de ferramentas de programação científica:

- álgebra linear (scipy.linalg)
- integração numérica (scipy.integrate)
- interpolação (scipy.interpolate)
- otimização (scipy.optimize)
- estatística (scipy.stats)
- outras

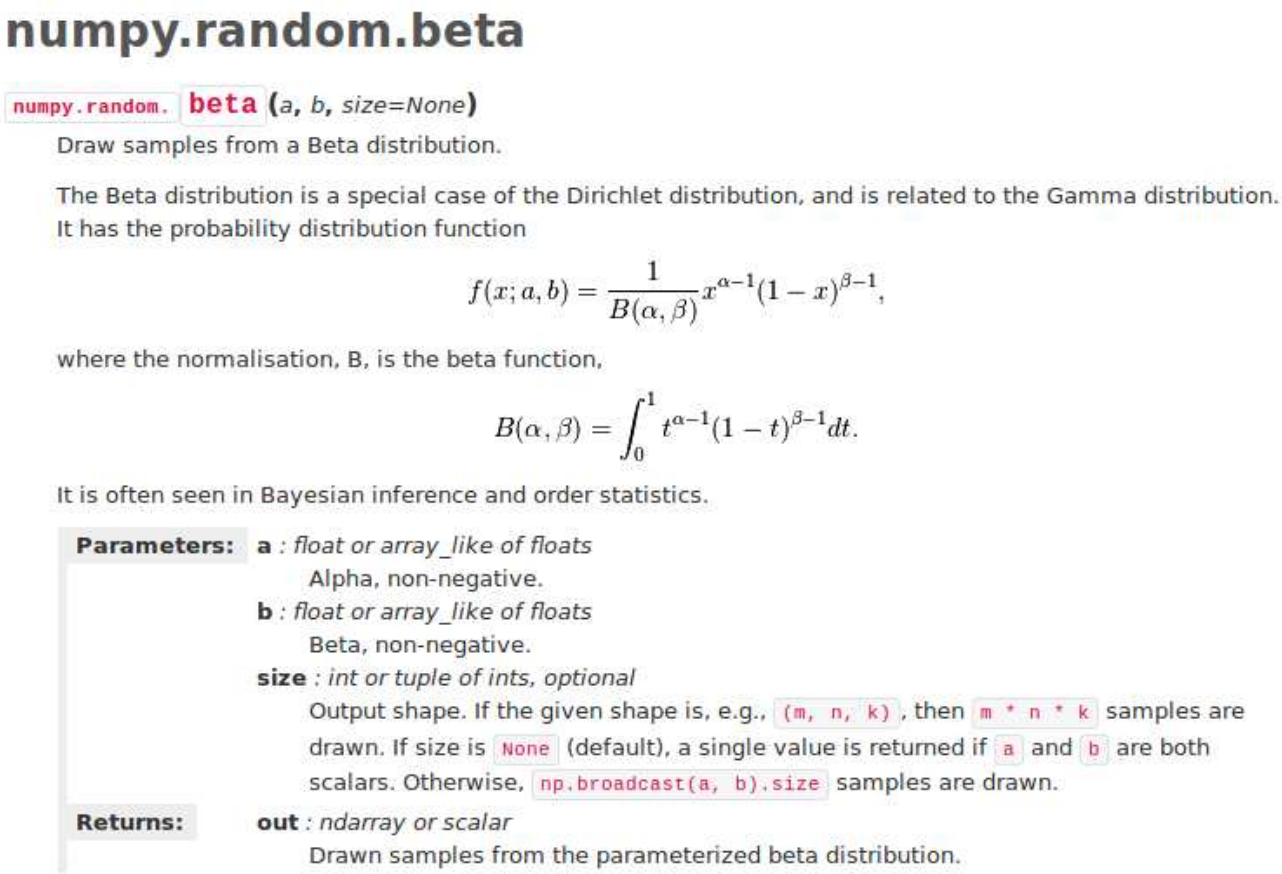
Scipy - Estatística - scipy.stats

Este pacote oferece:

- Variáveis aleatórias: densidade, distribuições, amostragem aleatória, etc.
- procedimentos de estimativa
- testes estatísticos
- outros
- Veja a lista completa de funções deste módulo utilizando o comando *help* no *Spider* ou consultando a documentação na internet (https://docs.scipy.org/doc/scipy/reference/stats.html).

Scipy - Estatística - scipy.stats

Considere que queremos usar a *função Beta*. Podemos usar a função do Numpy:



numpy.random.beta

`numpy.random.` **`beta`** (*a*, *b*, *size=None*)

Draw samples from a Beta distribution.

The Beta distribution is a special case of the Dirichlet distribution, and is related to the Gamma distribution. It has the probability distribution function

$$f(x; a, b) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1},$$

where the normalisation, B, is the beta function,

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt.$$

It is often seen in Bayesian inference and order statistics.

Parameters:

a : *float or array_like of floats*
Alpha, non-negative.

b : *float or array_like of floats*
Beta, non-negative.

size : *int or tuple of ints, optional*
Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. If size is `None` (default), a single value is returned if `a` and `b` are both scalars. Otherwise, `np.broadcast(a, b).size` samples are drawn.

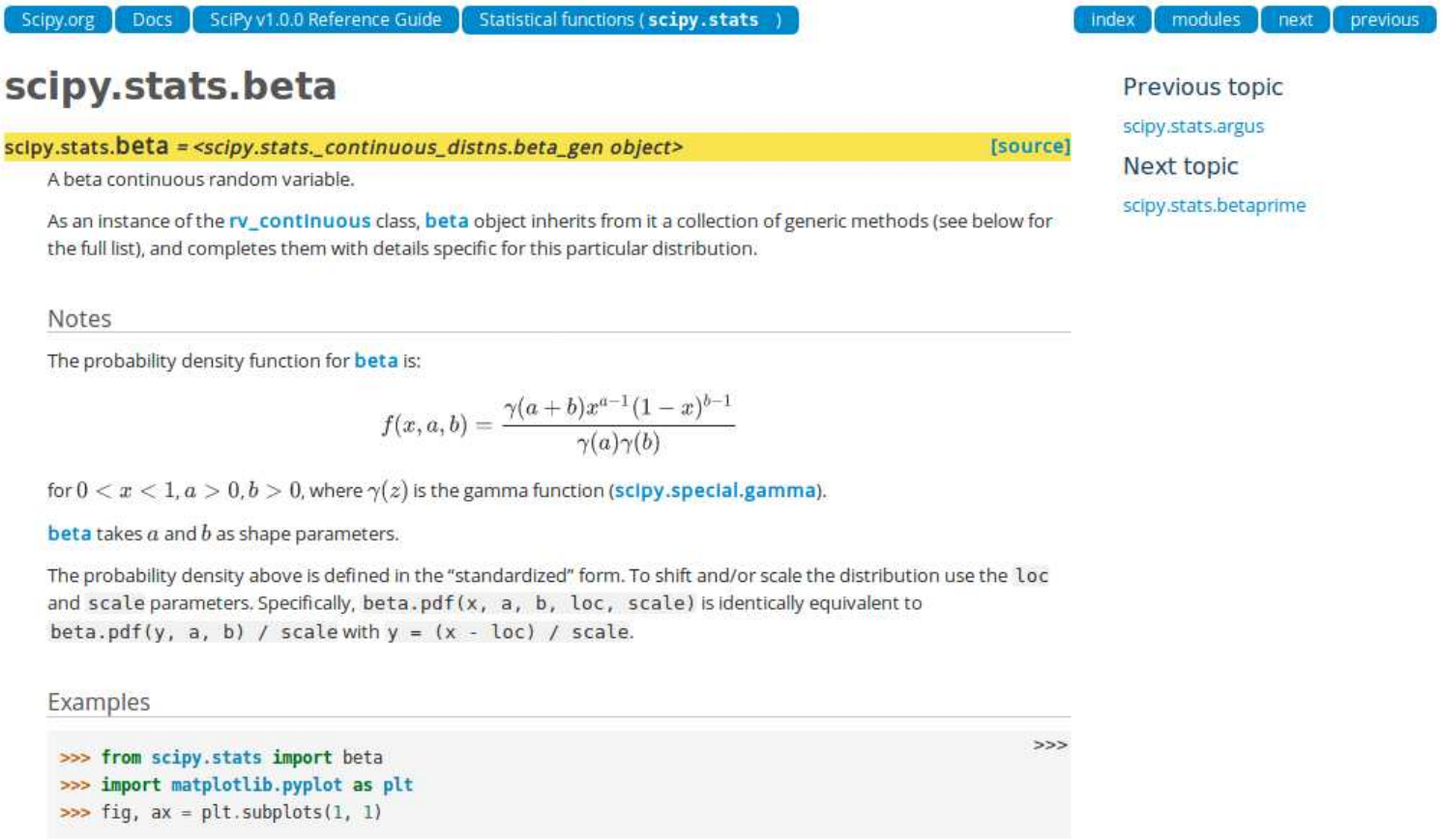
Returns:

out : *ndarray or scalar*
Drawn samples from the parameterized beta distribution.

https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.beta.html

Scipy - Estatística - scipy.stats

Ou a função do Scipy.



Scipy.org | Docs | SciPy v1.0.0 Reference Guide | Statistical functions (**scipy.stats**) | index | modules | next | previous

scipy.stats.beta

scipy.stats.beta = *<scipy.stats._continuous_distns.beta_gen object>* [source]

A beta continuous random variable.

As an instance of the **rv_continuous** class, **beta** object inherits from it a collection of generic methods (see below for the full list), and completes them with details specific for this particular distribution.

Notes

The probability density function for **beta** is:

$$f(x, a, b) = \frac{\gamma(a+b)x^{a-1}(1-x)^{b-1}}{\gamma(a)\gamma(b)}$$

for $0 < x < 1, a > 0, b > 0$, where $\gamma(z)$ is the gamma function (**scipy.special.gamma**).

beta takes a and b as shape parameters.

The probability density above is defined in the "standardized" form. To shift and/or scale the distribution use the `loc` and `scale` parameters. Specifically, `beta.pdf(x, a, b, loc, scale)` is identically equivalent to `beta.pdf(y, a, b) / scale` with `y = (x - loc) / scale`.

Examples

```
>>> from scipy.stats import beta
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Previous topic: scipy.stats.argus

Next topic: scipy.stats.betaprime

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.beta.html#scipy.stats.beta

Scipy - Estatística - scipy.stats

O Scipy nos oferece muito mais recursos:

Methods

`rvs(a, b, loc=0, scale=1, size=1, random_state=None)`	Random variates.
`pdf(x, a, b, loc=0, scale=1)`	Probability density function.
`logpdf(x, a, b, loc=0, scale=1)`	Log of the probability density function.
`cdf(x, a, b, loc=0, scale=1)`	Cumulative distribution function.
`logcdf(x, a, b, loc=0, scale=1)`	Log of the cumulative distribution function.
`sf(x, a, b, loc=0, scale=1)`	Survival function (also defined as `1 - cdf`, but *sf* is sometimes more accurate).
`logsf(x, a, b, loc=0, scale=1)`	Log of the survival function.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.beta.html#scipy.stats.beta

Scipy - Estatística - scipy.stats

- **norm**: variável aleatória contínua normal
- Função de Densidade da Distribuição Normal

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

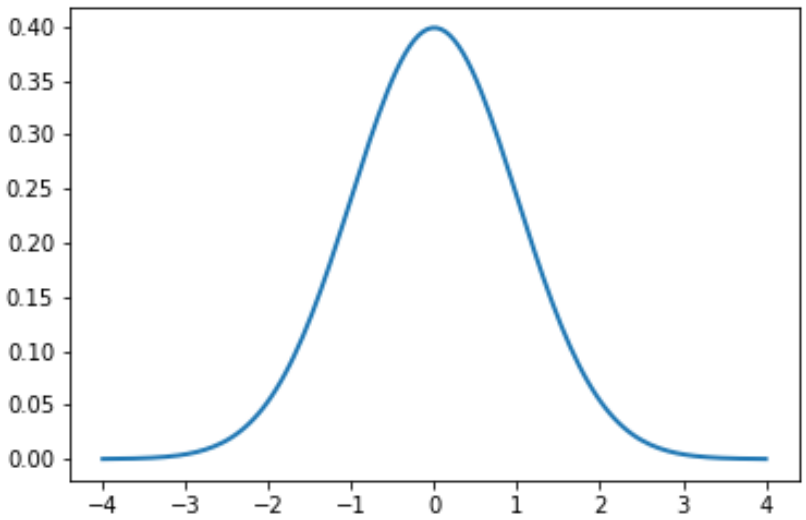
  onde μ é a média e σ o desvio padrão.
- No scipy, ela é definida para a $\mu = 1$ (parâmetro *loc*) e $\sigma = 0$ (parâmetro *scale*) por default:

$$f(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$$

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import norm # variavel aleatoria continua normal
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = norm.pdf(x)
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('normal1.png')
```

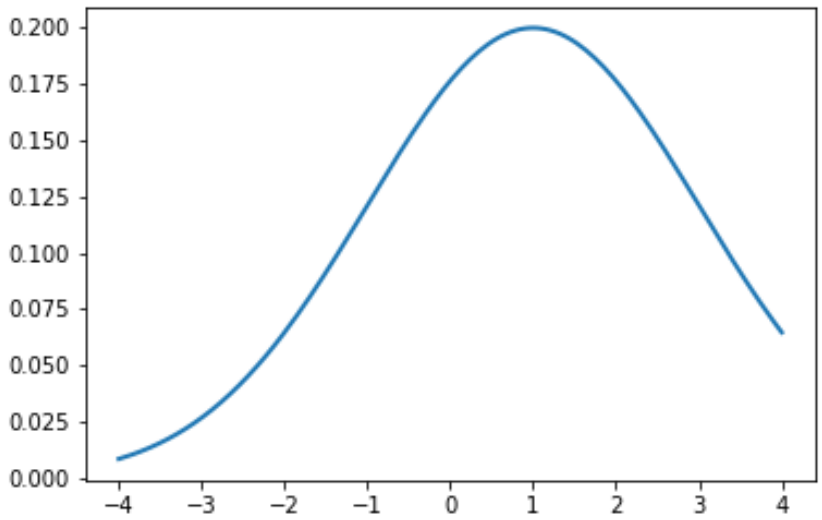
Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import norm # variavel aleatoria continua normal
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = norm.pdf(x)
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('normal1.png')
```



Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import norm # variavel aleatoria continua normal
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = norm.pdf(x,loc=1,scale=2) # media = 1 e desvio = 2
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('normal2.png')
```



Scipy - Estatística - scipy.stats

- **uniform**: variável aleatória contínua uniforme
- Função de Densidade da Distribuição Uniforme

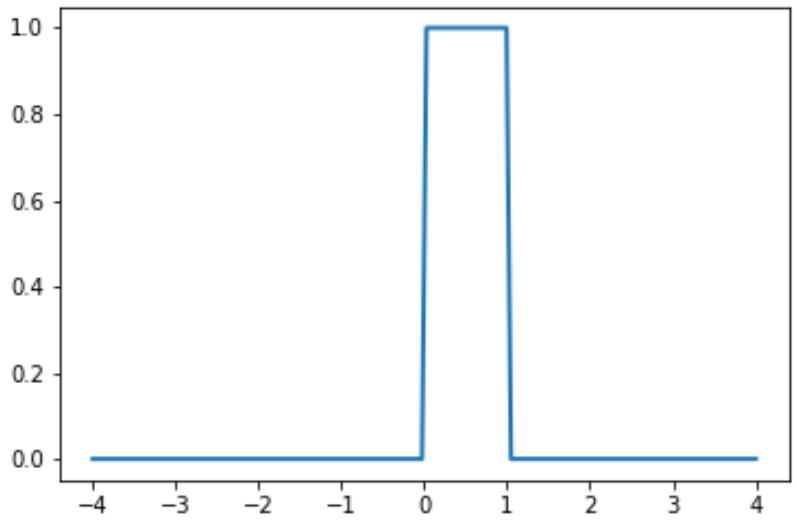
$$f(n) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b \\ 0 & \text{caso contrário} \end{cases}$$

- No scipy, o intervalo $[a, b]$ é definido por $[loc, loc + scale]$, onde por default $loc = 0$ e $scale = 1$:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import uniform # variavel aleatoria continua uniforme
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = uniform.pdf(x)  # densidade uniforme
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('uniforme1.png')
```

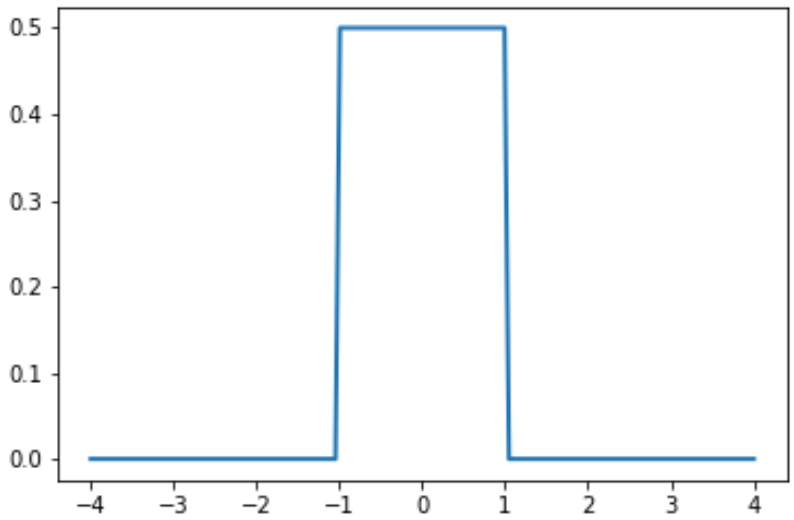
Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import uniform # variavel aleatoria continua uniforme
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = uniform.pdf(x)  # densidade uniforme
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('uniforme1.png')
```



Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy.stats import uniform # variavel aleatoria continua uniforme
4
5
6  fig, ax = plt.subplots()
7
8  x = np.linspace(-4, 4, 150)
9  y = uniform.pdf(x,loc=-1,scale=2) # mudando o intervalo
10
11 ax.plot(x, y, linewidth=2)
12
13 plt.savefig('uniforme2.png')
```



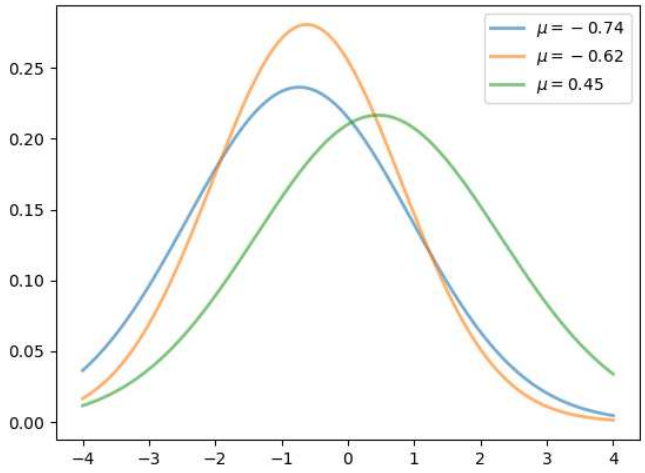
Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  from scipy.stats import norm    # 1
5  from random import uniform      # 2
6
7  fig, ax = plt.subplots()
8  x = np.linspace(-4, 4, 150)
9  for i in range(3):    # 3
10     m, s = uniform(-1, 1), uniform(1, 2)
11     y = norm.pdf(x, loc=m, scale=s)  # 4
12     current_label = r'$\mu = {0:.2f}$'.format(m)
13     ax.plot(x, y, linewidth=2, alpha=0.6, label=current_label)
14 ax.legend()
15 plt.show()
```

1. **norm**: Variável aleatória contínua normal
2. **uniform**: sorteia um número float dentro de um dado intervalo. **CUIDADO: não é do scipy, é do random!**
3. Em cada passada do loop, um gráfico é construído:
   - Sorteia uma média $m \in [-1., 1.]$ e desvio padrão $s \in [1., 2.]$
   - **norm.pdf**: função densidade de probabilidade com média m (loc=m) e desvio padrão s (scale=s)

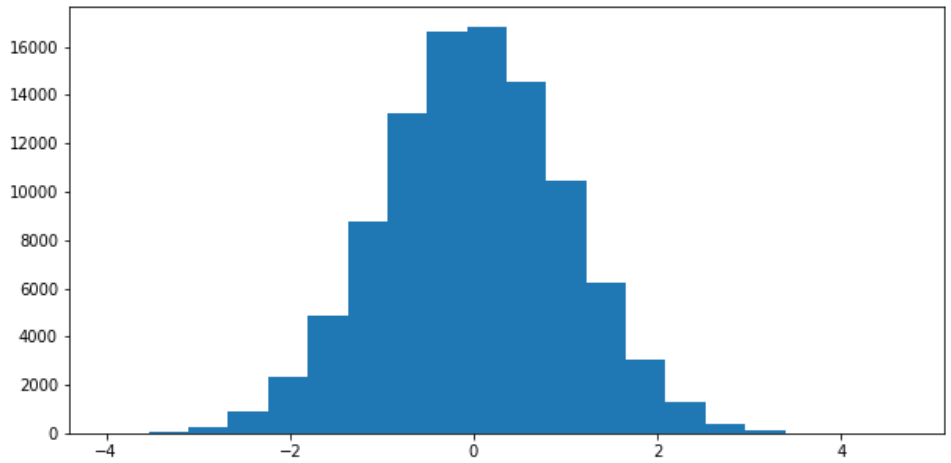
Scipy - Estatística - scipy.stats

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  from scipy.stats import norm    # 1
5  from random import uniform      # 2
6
7  fig, ax = plt.subplots()
8  x = np.linspace(-4, 4, 150)
9  for i in range(3):    # 3
10     m, s = uniform(-1, 1), uniform(1, 2)
11     y = norm.pdf(x, loc=m, scale=s)  # 4
12     current_label = r'$\mu = {0:.2f}$'.format(m)
13     ax.plot(x, y, linewidth=2, alpha=0.6, label=current_label)
14 ax.legend()
15 plt.show()
```



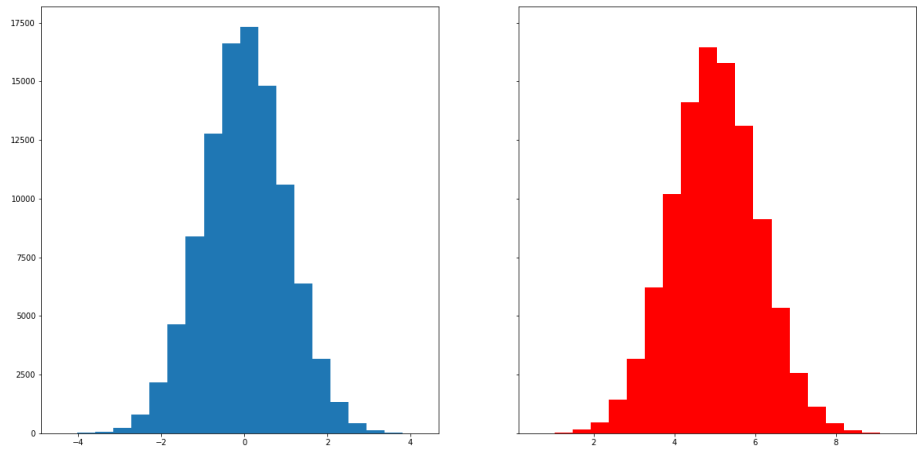
Matplotlib - Histogramas

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  numeroPontos = 100000 # numero de pontos
5  numeroBins = 20    # numero de colunas
6
7  x = np.random.randn(numeroPontos)    # eixo X
8  y = .4 * x + np.random.randn(numeroPontos) + 5  # eixo Y
9
10 fig, axs = plt.subplots(figsize=(10,5))
11
12 axs.hist(x, bins=numeroBins)  # histograma
13
14 plt.savefig('teste0.png')
```



Matplotlib - Histogramas

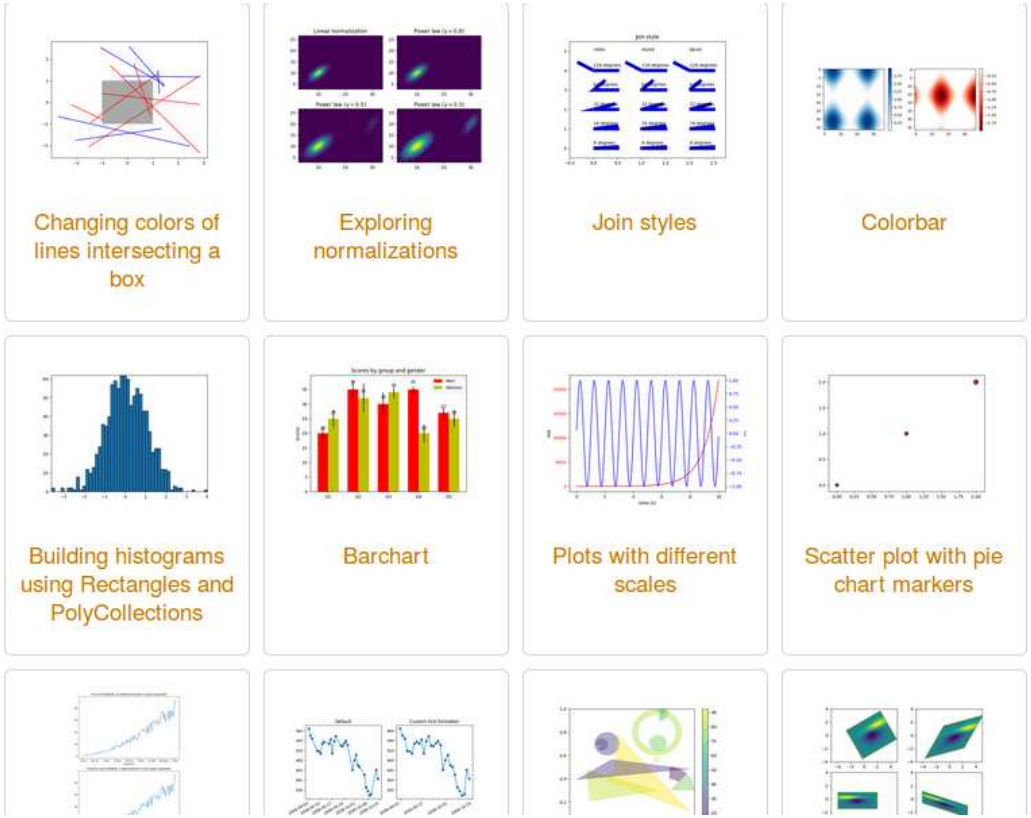
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  numeroPontos = 100000
5  numeroBins = 20
6
7  x = np.random.randn(numeroPontos)   # eixo X
8  y = .4 * x + np.random.randn(numeroPontos) + 5  # eixo Y
9
10 fig, axs = plt.subplots(1, 2, figsize=(20,10),sharey=True) # dois histog. lado a lado com
       mesmo eixo Y (sharey)
11
12 axs[0].hist(x, bins=numeroBins)   # primeiro histograma
13 axs[1].hist(y, bins=20,facecolor='r') # segundo histograma, de cor vermelha
14
15 plt.savefig('teste.png')
```



Matplotlib - Galeria

A Matplotlib é uma biblioteca bastante poderosa. Existem inúmeros modelos de gráficos que podem ser usados.

Veja em : https://matplotlib.org/gallery/index.html



Matplotlib - Referências

- Quantecon: https://lectures.quantecon.org/py/matplotlib.html#the-matlab-style-api
- Matplotlib: https://matplotlib.org/users/pyplot_tutorial.html

Autores

- João C. P. da Silva ▸ Lattes
- Carla Delgado ▸ Lattes

Computação II - Python
Aula 8 - Matplotlib e Scipy

João C. P. da Silva

Carla A. D. M. Delgado

Dept. Ciência da Computação - UFRJ