Gráficos - Matplotlib

Python tem uma poderosa biblioteca para produção de gráficos de boa qualidade: **Matplotlib**. Para gráficos simples, podemos usar o módulo **pyplot** que deve ser importado da seguinte forma:

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

No jupyter, para que o gráfico apareça numa célula do notebook, digite na primeira célula: `%matplotlib inline`. Para que o gráfico seja mostrado numa janela separada, digite na primeira célula: `%matplotlib`.

Gráficos - Gráfico de Funções

Se quisermos fazer um gráfico de uma função, as entradas para o pyplot devem ser arrays (ou listas) correspondentes aos valores x e y. Exemplo:

Exemplo 1 - Gráfico simples

```
x = np.linspace(-3*np.pi,3*np.pi,100)
y = np.sin(x)    #vetorização
plt.plot(x,y)
```

Para adicionar um segundo plot, basta chamar `plt.plot` novamente:

```
z = np.cos(x)
plt.plot(x,z)    #ou plt.plot(x,y,x,z)
```

Gráficos - Legenda

Para nomear um gráfico, devemos atribuir um string ao argumento `label` da função `plot`. Para adicionar a legenda no gráfico, faça:
`plt.legend()`

Exemplo 2 - Legenda

```
x = np.linspace(-3*np.pi,3*np.pi,100)
y = np.sin(x)
z = np.cos(x)
plt.plot(x,y,label='sen(x)')
plt.plot(x,z,label='cos(x)')
plt.legend()
```

Para retirar a legenda da "caixa", use a opção `frameon=False`. Para selecionar o tamanha da fonte, use `fontsize=<inteiro>`.

Gráficos - Legenda

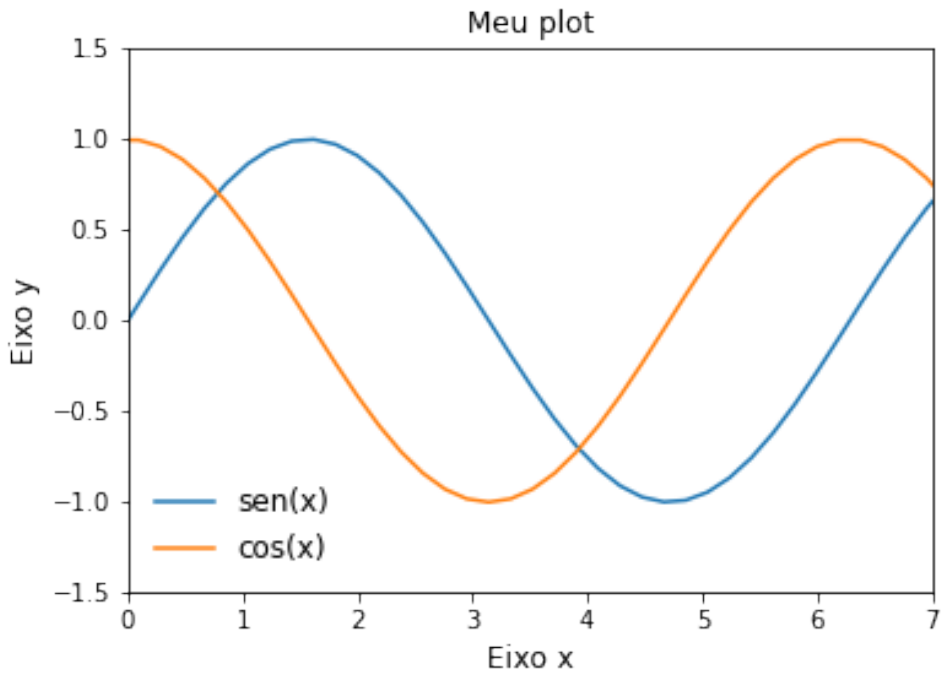
Opções de localização da legenda

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

Gráficos - Descrição dos Eixos e Título

Exemplo 3 - Eixos e Título

```
plt.title('Meu plot')
plt.xlabel('Eixo x',fontsize=12)
plt.ylabel('Eixo y',fontsize=12)
plt.xlim(0, 7.)      #x-range
plt.ylim(-1.5,1.5)   #y-range
```



Gráficos - Marcadores, Cores e Linhas

Existem diversas opções de marcadores, linhas e cores, que devem ser especificados por strings. Por exemplo, se quisermos linha vermelha tracejada, basta incluir 'r- -' na função plot.

Exemplo 3 - Cores e Linhas

```
plt.plot(x,y,'r--',label='sen(x)')
plt.plot(x,y,'r--o',label='sen(x)')  #marcador 'o'
```

Também é possível passar os atributos explicitamente com `c` (color), `marker` (marcador) `ls` (estilo da linha) e `lw` (largura da lilnha)

```
plt.plot(x,y, c='r',marker='o',ls='--',lw=2)
```

Também é possível selecionar o tamanho do marcador (`markersize`), a cor (`markerfacecolor` ou `mfc`), e a cor da borda (`markeredgecolor` ou `mec`).

Gráficos - Marcadores, Cores e Linhas

Marcadores

Código	Marcador
`.`	Ponto
`o`	Círculo
`+`	Cruz
`x`	Cruzado
`D`	Diamante
`v`	Triângulo p/ baixo
`^`	Triângulo p/ cima
`s`	Quadrado
`*`	Estrela

Estilos de linha: (-) (–) (:) (-.)

Cores Básicas

Código	Cor
`r`	Vermelho
`g`	Verde
`b`	Azul
`c`	ciano
`m`	magenta
`y`	Amarelo
`k`	Preto
`w`	Branco
`brown`	Marrom
`gray`	Cinza
`purple`	Roxo

Gráfico de Barras

Gráficos de barras são feitos com a função `plt.bar`.

Example

```
anos = np.arange(2014,2021)
consumo = np.array([327,392,490,643,806,981,1171])
plt.bar(anos, consumo, width=0.6,color='g',
        edgecolor=None)
plt.xlabel('Anos')
plt.ylabel('Consumo em milhares de sacas')
```

Gráficos - Dois Eixos

O comando `plt.twinx()` cria um novo eixo y mantendo o mesmo eixo x (também existe a opção plt.twiny()).

Example

```
line1 = plt.plot(tempo,divorcios,'b-o')
plt.ylim(4,5.2)
plt.ylabel('Divorcios por 100 mil')
plt.xlabel('Anos')
plt.twinx()
line2 = plt.plot(tempo,margarina,'r-v')
plt.ylabel('Consumo de Margarina [lb]')
lines = line1 + line2
legendas = ['Divorcios','Consumo de Margarina']
plt.legend(lines, legendas, frameon=False)
```

Gráfico Com Barras de Erro

Gráficos com barras de erro podem ser criados com a função `plt.errorbar`. Pode-se escolher várias opções de formataçã para a barra de erro.

Example

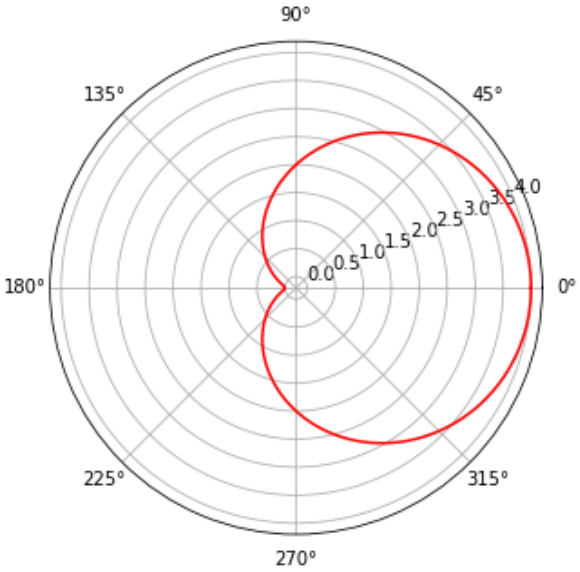
```
x = np.linspace(0,10,30)
dy = 0.8
y = np.sin(x) + dy*np.random.random(30)
#plt.errorbar(x,y,yerr=dy,fmt='ko')
plt.errorbar(x,y,yerr=dy,ecolor='gray',elinewidth=2,
             capsize=3,fmt='ko')
```

Matplotlib - Gráficos Polares

Gráficos polares (raio r em função de um ângulo θ) podem ser criados usando a função `pyplot.polar()`. Como exemplo, vamos fazer o gráfico de $r = 2a(1 + cos\theta)$ (cardioide).

Example

```
theta = np.linspace (0, 2*np.pi, 1000)
r = 2 * (1. + np.cos(theta ))
plt.polar(theta,r,'r')
```



Matplotlib - Histogramas

Histograma é uma representação gráfica de uma distribuição discreta de probabilidade. Para plotar um histograma, basta passar um array para a função `plt.hist()`. O exemplo abaixo ilustra as várias opções disponíveis.

Example

```
data = np.random.randn(1000)
plt.hist(data, bins=30, density=True, alpha=0.5,
         histtype='bar', color='steelblue')
```

Para obter a contagem em cada bin, podemos usar o método `np.histogram()`:

Example

contagem, x_bin = np.histogram(data,bins=10)

Matplotlib - Interface Avançada

O Matplotlib tem uma interface básica, semelhante ao MATLAB, que permite fazer vários gráficos simples (vejas exemplos nos slides anteriores). No entanto, para ter maior controle sobre os elementos do gráfico, existe uma interface orientada a objetos. Nessa interface, criamos um objeto chamado *figure*, que pode ser pensando como um contêiner que contém todos os objetos relacionados aos eixos, gráficos e descrições. Acoplamos ao *figure* o objeto *axes*, que contém todos os elementos do gráfico.

Criando uma Figura

```
fig = plt.figure()
ax = fig.add_subplot()
```

Também é possível criar os dois objetos numa única linha

```
fig, ax = plt.subplots()
```

Matplotlib - Interface Avançada

O objeto *figure* tem vários argumentos opcionais, como identificador e tamanho da figura.

Argumento	Descrição
`num`	String identificador da figura
`figsize`	Tupla com as dimensões da figura (largura, comprimento), em polegadas
`dpi`	Resolução da figura (pts por polegada)
`facecolor`	Cor de fundo
`edge color`	Cor da borda

Example

```
fig = plt.figure("Figura1", figsize=(4.5,3),
                 facecolor='r')
```

Matplotlib - Fontes

Os elementos de texto de um plot podem ser modificados com as opções da tabela abaixo.

Argumento	Descrição
`fontsize`	Tamanho da fonte
`fontname`	Nome (ex. 'Arial')
`family`	Família (ex. 'cursive')
`fontweight`	Peso (ex.'normal', 'bold')
`fontstyle`	Estilo (ex.'normal', 'italic')
`color`	Cor da fonte

Para usar a mesmas opções de fonte em todos os textos (título, labels), podemos construir um dicionário e usar o comando `rc`.

Matplotlib - Fontes

Parâmetros da Fonte

```
from matplotlib import rc
font_properties = {'name': 'Courier',
                   'weight': 'bold ', 'size': 13}

rc('font ', **font_properties )
```

Matplotlib - Gridlines e Escala Log

Podemos adicionar linhas de grade aos eixos vertical e/ou horizontal. É possível escolher o estilo da linha e cor (`linestyle`, `linewidth`, `color`, etc.).

Gridlines

```
ax.grid(True) #linhas horizontais e verticais
ax.yaxis.grid(True) #apenas horizontal
```

Escala logarítmica pode ser escolhida com os comandos abaixo. Por padrão, logaritmo decimal é usado, mas podemos escolher outra base com os argumentos `basex`, `basey`.

Escala Log

```
ax.set_xscale('log')
ax.set_yscale('log')
```

Matplotlib - Marcadores

Existem diversas opções para modificar marcadores no Matplotlib. Por exemplo, para definir marcadores para apenas algums valores do eixo, fazemos:
`ax.set_yticks([1, 2, 3.5, 4.])`.
É possível substituir os marcadores numéricos por *strings* usando os comandos `ax.set_xticklabels` e `ax.set_yticklabels`.

Matplotlib - Marcadores

Esconder Marcadores

`ax.set_yticks([])`: apaga marcadores e labels.

`ax.set_yticklabels([])`: apaga os labels, mantém os marcadores.

Adicionar submarcadores

`ax.minorticks_on()`

Adicionar marcadores aos eixos paralelos

`ax.xaxis.set_ticks_position('bottom')`

As opções são `'bottom'`, `'top'`,`'both'`, ou `'none'`

`ax.yaxis.set_ticks_position('both')`

As opções são `'left'`, `'right'`,`'both'`, ou `'none'`

Matplotlib - Marcadores

Opções mais avançadas de marcadores estão disponíveis na função `ax.tick_params()`, com os argumentos da tabela abaixo.

Argumento	Descrição
`axis`	Eixo que será alterado: 'x','y','both'
`which`	Marcador:'major','minor', 'both'
`direction`	'in', 'out','inout
`length`	Comprimento do marcador
`width`	Largura do marcador
`labelsize`	Tamanho do label
`color`	Cor do marcador
`labelcolor`	Cor do label
`labeltop/labelright`	True ou False

Matplotlib - Subplots

Subplots são grupos de gráficos que pertencem a uma mesma figura. Existem diferentes formas de criar subplots no Matplotlib.

Métodos plt.subplots()

```
fig, axes = plt.subplots(nrows=3, ncols=2)
fig.tight_layout()
```

Será criada uma figura com seis gráfico (3 x 2), e cada um pode ser acessado com índices semelhantes a elementos de matrizes:

```
ax1 = axes[0, 0]   #superior esquerdo
ax2 = axes[2, 1]   #inferior direito
```

É possível escolher o tamanho da figura passando para a função `plt.subplots()` o argumento `figsize=(<int>,<int>)`.

Matplotlib - Subplots

Para ajustar a distância entre os gráficos, usamos `fig.subplots_adjust(hspace=<float>, wspace=<float>)`.

Example

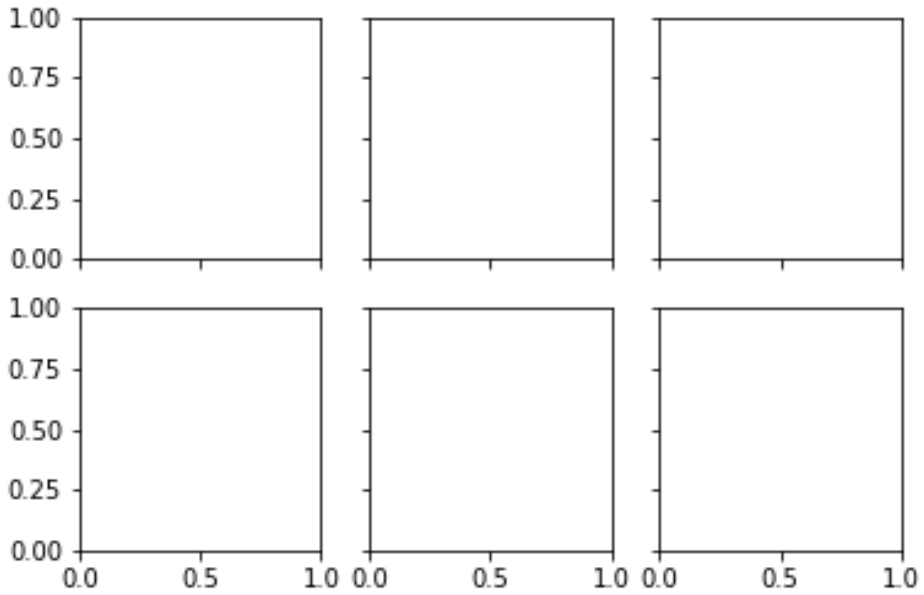
```
fig, axes = plt.subplots(2,1)
fig.subplots_adjust(hspace=0.05)
x = np.linspace(0,2*np.pi,100)
axes[0].plot(x,np.sin(x))
axes[0].set_xticks([])
axes[1].plot(x,np.cos(x))
```

Matplotlib - Subplots

Podemos ocultar automaticamente os labels internos dos gráficos usando os argumentos `sharex` e `sharey` da função `plt.subplots()`.

Example

```
fig, ax = plt.subplots(2, 3, sharex='col',
                       sharey='row')
```



Matplotlib - Subplots

Em algumas situações, é conveniente fazer um gráfico menor dentro de um gráfico maior, para realçar um resultado. Esse tipo de figura pode ser feito com o método `plt.axes()`

Métodos plt.axes()

```
fig = plt.figure()
ax1 = plt.axes()
ax2 = plt.axes([0.65,0.65,0.2,0.2])
```

Os primeiros dois valores indicam que a posição do segundo plot começa em 65% da largura e 65% do comprimento de ax1, e seu tamanho é de 20% de ax1.

Matplotlib - Scatter Plots

A função pyplot.scatter() permite criar gráficos de dispersão onde as propriedades de cada ponto (cor, tamanho) podem ser controladas. Além dos valores x e y, podemos passar uma sequência de valores para os argumentos `s` e `c`, que controlam o tamanho e a cor da cada ponto. Esse tipo de gráfico é útil para visualizar dados multidimensionais.

Example

```
x = np.random.randn(100)
y = np.random.randn(100)
colors = np.random.rand(100)
sizes = 1000*np.random.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3)
plt.colorbar()
```

Matplotlib - Anotações

O método `ax.text(x,y,s)` permite incluir um *string* `s` na posição (x,y) dos **dados**. Para fixar o texto numa determinada posição independente do range dos dados, devemos usar o argumento `transform=ax.transAxes`. Nesse caso, a coordenada (0,0) representa o canto inferior esquerdo do eixo, e (1,1) o canto superior direito.

Example

```
fig, ax = plt.subplots(figsize=(6,6))
x = np.linspace(0,5)
ax.plot(x, x, 'o')
ax.text(1,3,'Aqui',fontsize=14)
```

Matplotlib - Exemplo de Aplicação

Vamos escrever um código em Python para ler o arquivo `higgs_data.csv` e fazer um gráfico dos dados e de duas funções, chamadas de *background* e sinal. A função background é dada por

$$f_{bkg} = ax^3 + bx^2 + cx + d$$

onde d, c, b, a são dados, nessa ordem, pela lista [24415.1,-356.488,1.80183,-0.00307196]. A função sinal é dada por

$$f_{sig} = A \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

onde A, μ e σ valem, nessa ordem, [200,126.218,2.3875].
Referência: Figura 4 do artigo PL **B716** 2012

Matplotlib - Gráficos de Contorno

Uma função $h(x, y)$ pode ser interpretada como uma superfície em que cada ponto (x, y) tem um altura específica. Um gráfico de contorno é um gráfico em duas dimensões da função $h(x, y)$ em que linhas ou cores são usadas para representar a altura. Para fazer um gráfico desse tipo, precisamos especificar uma altura Z e um conjunto de pontos que formem uma grade no plano xy. Essa grade de pontos pode ser criada usando o método `numpy.meshgrid()`.

Example

```
x = np.linspace(-3,3,21)
y = np.linspace(0,10,11)
X,Y = np.meshgrid(x,y)
```

Os novos arrays X e Y tem forma 11×21 e $11 \times 21 = 231$ entradas.

Matplotlib - Gráficos de Contorno

Método `contour()`

Os argumentos são os 2D arrays X, Y e Z. Os níveis de contorno pode ser especificado por um número total `N` ou uma sequência que especifique os valores de Z que devem ser plotados. Para plotar contorno com um única cor, usamos o argumento `colors`.

Example

```
x = np.linspace(-3,3,21)
y = np.linspace(0,10,11)
X,Y = np.meshgrid(x,y)
Z =  np.sin(X)**10 + np.cos(10 + Y*X)*np.cos(X)
fig, ax = plt.subplots()
ax.contour(X,Y,Z,5,colors='k')
```

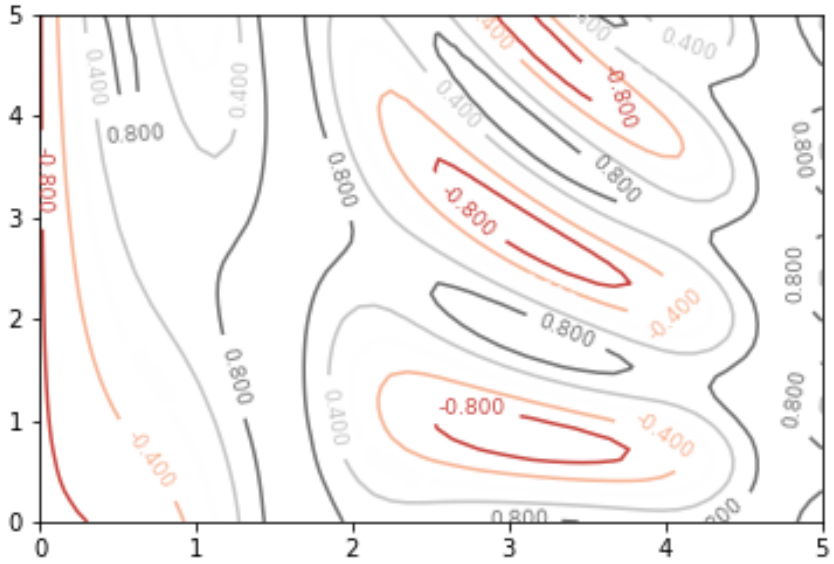
Cada nível de contorno também pode ser especificado por um mapa de cor com o argumento `cmap`. Exemplo: `cmap='RdGy'`.

Matplotlib - Gráficos de Contorno

Para adicionar os valores dos contornos (labels), usamos a função `ax.clabel()`. Nessa caso, o plot deve ser guardado numa variável e passado para a função `clabel()`.

Example

```
fig, ax = plt.subplots()
cp = ax.contour(X,Y,Z,5, cmap='RdGy')
ax.clabel(cp,fontsize=9)
```



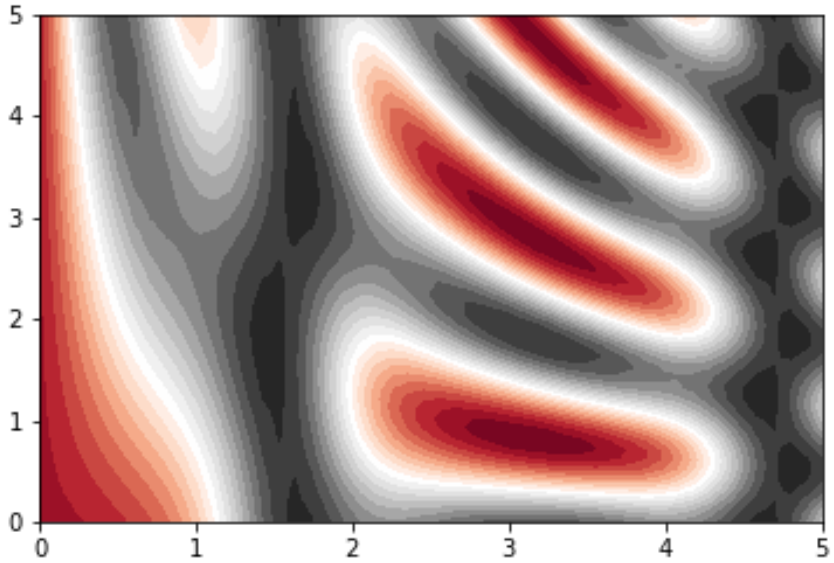
Matplotlib - Gráficos de Contorno

Método `contourf()`

O método `contourf()` tem os memos argumentos de `contour()`, mas produz contornos preenchidos.

Example

```
fig, ax = plt.subplots()
ax.contourf(X,Y,Z,20, cmap='RdGy')
```



Matplotlib - Mapa de Calor

Um mapa de calor é um imagem em que a cor de cada pixel é determinado por um valor correspondente numa sequência de dados. A função para produzir mapas de calor é `imshow()`. Alguns pontos importantes:

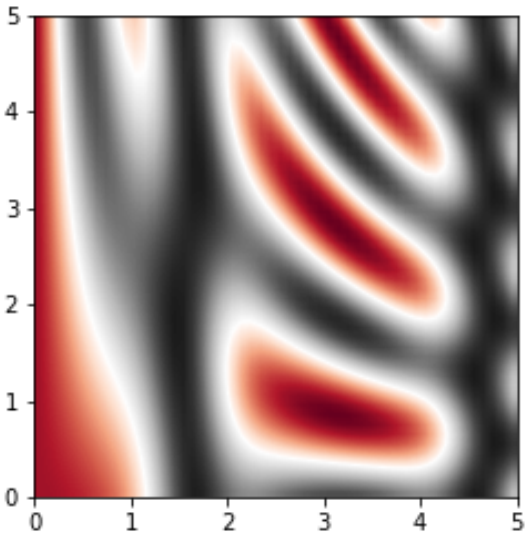
- `imshow()` não aceita um grid (x,y); o que é plotado nos eixos são os **índices** da matriz, sendo (0,0) o canto *superior* esquerdo.
- Para que a imagem seja mostrada nas coordenadas dos dados, devemos utilizar as opções `extent = [xmin, xmax, ymin, ymax]` e `origin='lower'`.
- pode-se aplicar interpolação com o argumento `interpolation`.

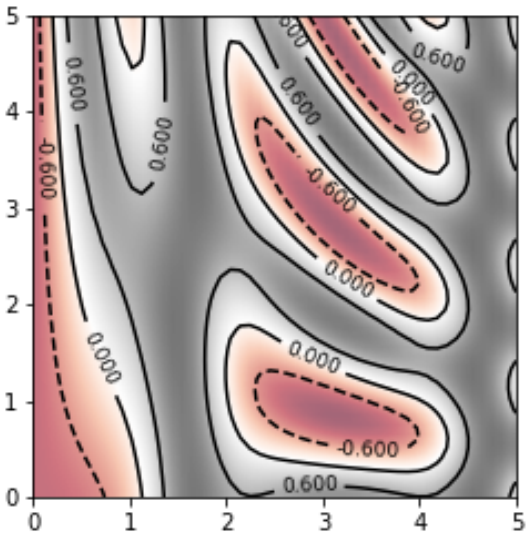
Matplotlib - Mapa de Calor

Podemos também combinar `imshow()` com `contour()`

Example

```
cp = ax.contour(X,Y,Z,3,colors='k')
ax.clabel(cp,fontsize=9)
ax.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
          interpolation='bilinear',alpha=0.6,
          cmap='RdGy')
```



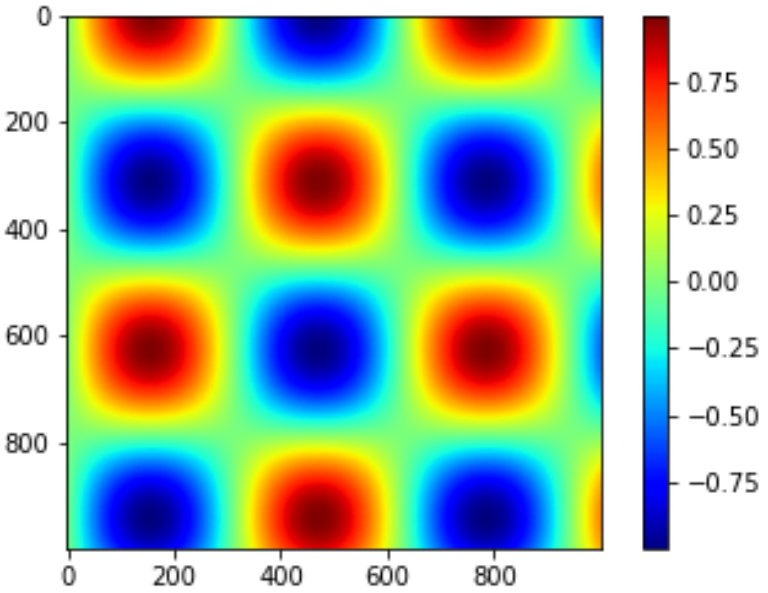


Matplotlib - *colorbar*

No Matplotlib, um *colorbar* é um eixo separado que indica como as cores do plot se relacionam aos valores do função $h(x, y)$. Para adicioná-lo a figura, chamamos o método `fig.colorbar(mappable = <obj>)`, onde `<obj>` é o objeto que guarda das informações do plot.

Example

```
x = np.linspace(0,10,1000)
y = np.linspace(0,10,1000)
X,Y = np.meshgrid(x,y)
I = np.sin(X) * np.cos(Y)
im= ax.imshow(I,cmap='jet')
fig.colorbar(im)
```

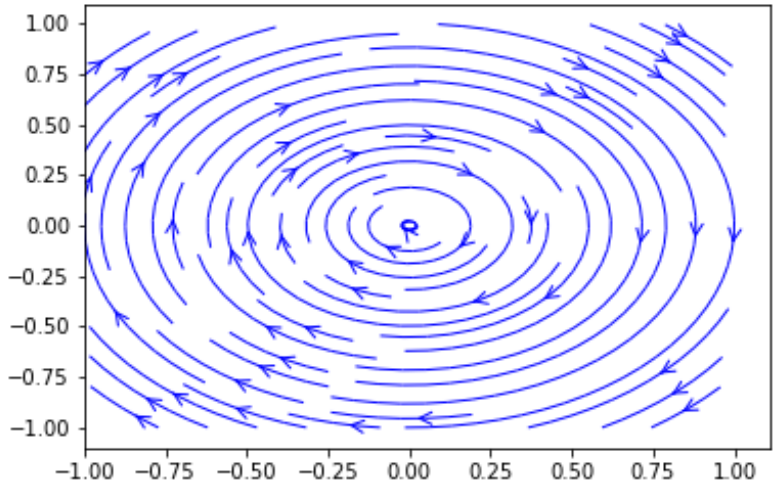


Matplotlib - Streamlines

Visualização de linhas de campo.

Example

```
xx = np.linspace(-1,1,11)
X,Y = np.meshgrid(xx,xx)
Vx,Vy = Y,-X
plt.streamplot(X,Y,Vx,Vy,color='b',
               linewidth=1, density=1,
               arrowstyle='->', arrowsize=1.5)
```



Matplotlib - Gráficos 3D

Para criar um plot tridimensional, devemos importar o função `Axes3D` a adicionar e argumento `projection =‘3d‘` ao subplot:

import Axes3D

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(projection ='3d')
```

Matplotlib - Gráficos 3D

Para plotar superfícies, devemos passar três arrays 2D. Os argumentos `rstride` e `cstride` indicam o passo em que linhas/colunas devem ser tomadas.

Wireframe e Superfícies

```
fig, ax = plt.subplots(figsize=(8,7),nrows=2,
          ncols=2, subplot_kw={'projection': '3d'})
ax[0,0].plot_wireframe(X, Y, Z, rstride=40,
                       cstride=40)
ax[0,1].plot_surface(X, Y, Z, rstride=40,
                     cstride=40, color='r')
ax[1,0].plot_surface(X, Y, Z, rstride=12,
                     cstride=12, color='m')
ax[1,1].plot_surface(X, Y, Z, rstride=20,
                     cstride=20, cmap='hot')
```