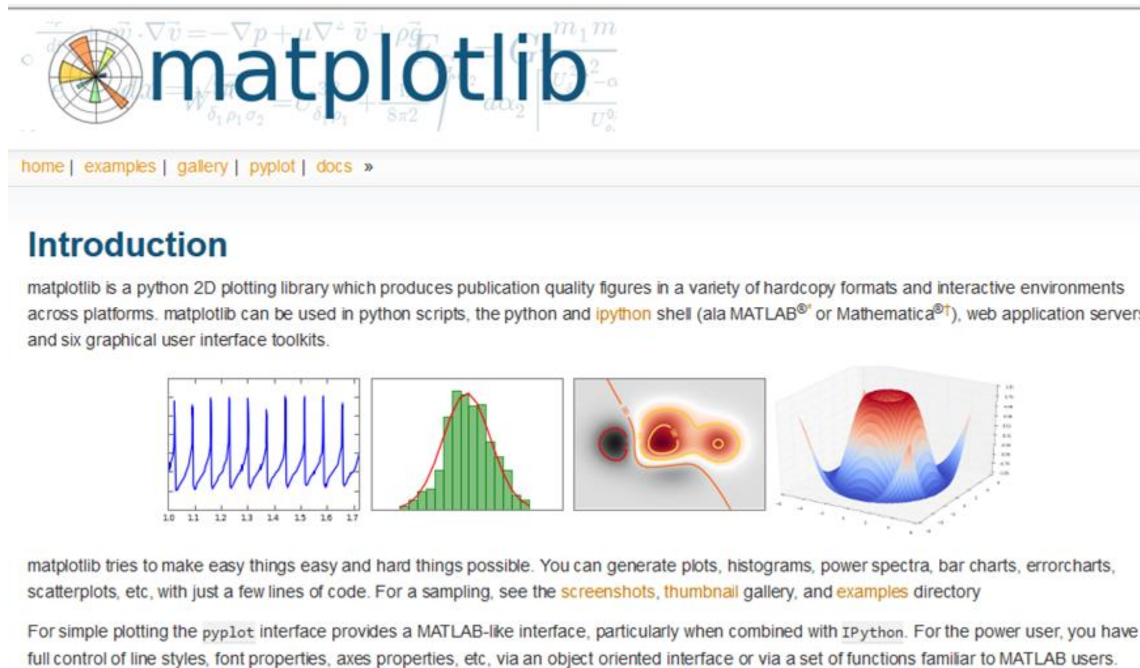
Bioinformática Aplicada

Aula 12

© 2019 Dr. Walter F. de Azevedo Jr.

python™
www.python.org

Uma das características da linguagem de programação Python, que tem atraído um grande número de fãs na comunidade de computação científica, incluindo bioinformática, é a possibilidade de uso de bibliotecas gratuitas para gráficos e análise numérica. Nas próximas aulas, introduziremos as bibliotecas *Matplotib*, *NumPy* e *SciPy*. Ao instalarmos o Python, por meio do *Pyzo*, já temos estas bibliotecas instaladas. Veremos alguns recursos das bibliotecas *Matplotlib* e *NumPy*.



Página de entrada do matplotlig: http://matplotlib.org/
Acesso em: 5 de junho de 2019.

Vejamos o uso das bibliotecas *Matplotlib* e *NumPy*, para gerarmos o gráfico do seno. A primeira linha de código importa a biblioteca *matplotlib* para gráfico, como *plt*. Assim, todos os recursos desta biblioteca podem ser chamados a partir da notação *dot* (.). O mesmo acontece na segunda linha de código, com a biblioteca *numpy*, importada como *np*. O método *.linspace()*, da biblioteca *numpy,* gera um conjunto de números *floats*, entre os intervalos mostrados, no caso, entre 0 e 2*Pi. O valor de Pi pode ser chamado com *numpy*, por meio de *np.py*. Dentro dos parênteses do método *linspace()*, o último número indica o número de pontos gerados, entre os valores máximo e mínimo.

```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine function
sine_func = np.sin(x)
# Creates plot
plt.plot(x, sine_func)
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('sine1.png')
```

Programa: *sine1.py*

python™
www.python.org

Os números gerados com o método *np.linspace(0,2*np.pi,100)* são atribuídos à variável *x*. Veja que o último número (100) indica o número de *floats* gerados no intervalo estabelecido, pelos dois primeiros números, dentro dos parênteses.
A linha de comando *sine_func = np.sin(x),* define a função seno, a partir da biblioteca *numpy*. O resultado é atribuído à variável *sine_func*. O método *plt.plot(x,sine_func)* gera o gráfico cartesiano, os valores do eixo x são aqueles atribuídos à variável *x*, e os valores do eixo y, são aqueles atribuídos à variável *sine_func*.

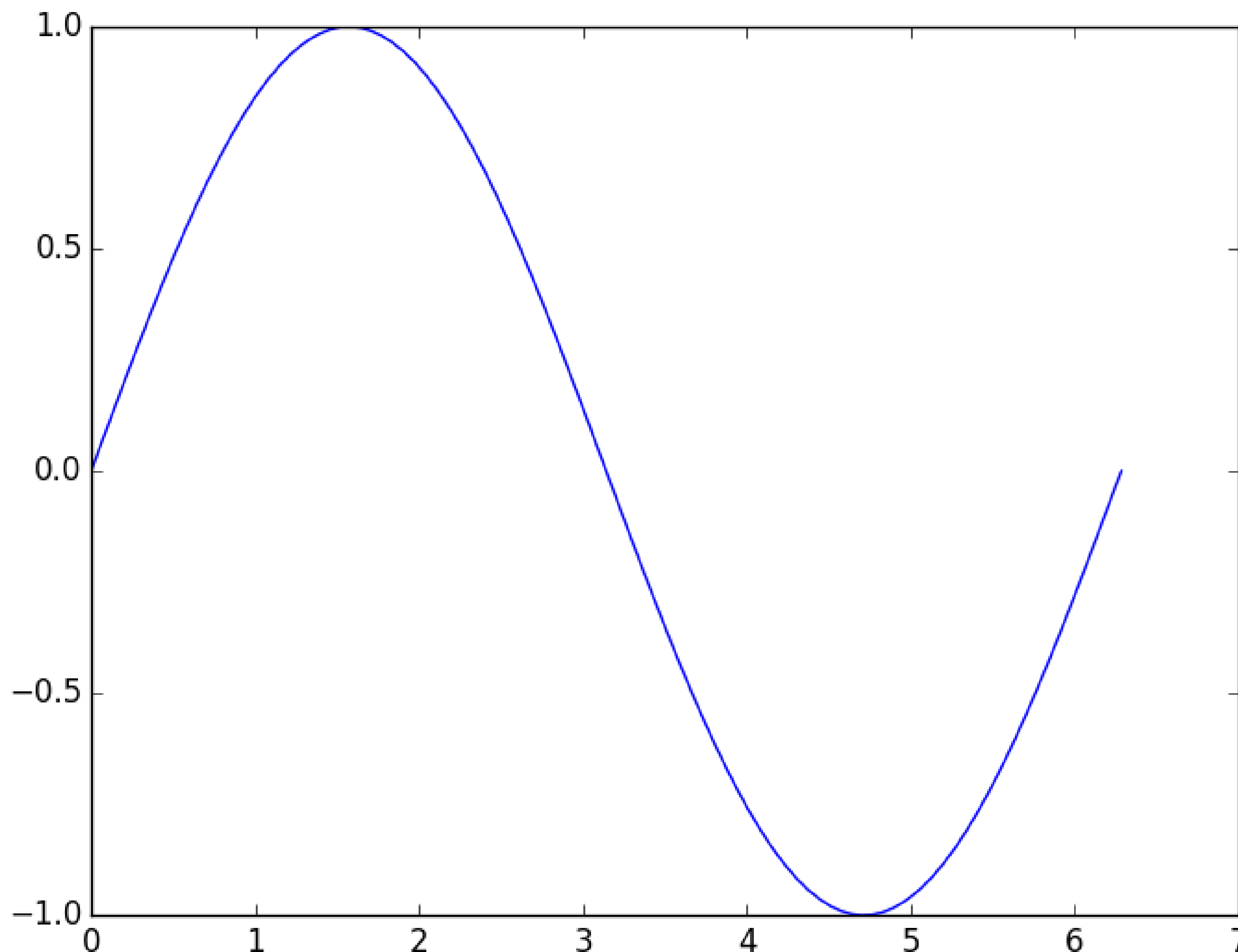
```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine function
sine_func = np.sin(x)
# Creates plot
plt.plot(x, sine_func)
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('sine1.png')
```

python™
www.python.org

O método *plt.show()* mostra o gráfico na tela. Este método pode ter resultados diferentes, conforme a instalação da biblioteca *Matplotlib* e o sistema operacional que você está utilizando. O método *plt.savefig('sine1.png')* salva o gráfico gerado no arquivo *sine1.png*. Execute o código *sine1.py* e clique na pasta da aula de hoje. Localize o arquivo *sine1.png* e clique duas vezes neste. Veja o gráfico gerado.

```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine function
sine_func = np.sin(x)
# Creates plot
plt.plot(x, sine_func)
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('sine1.png')
```

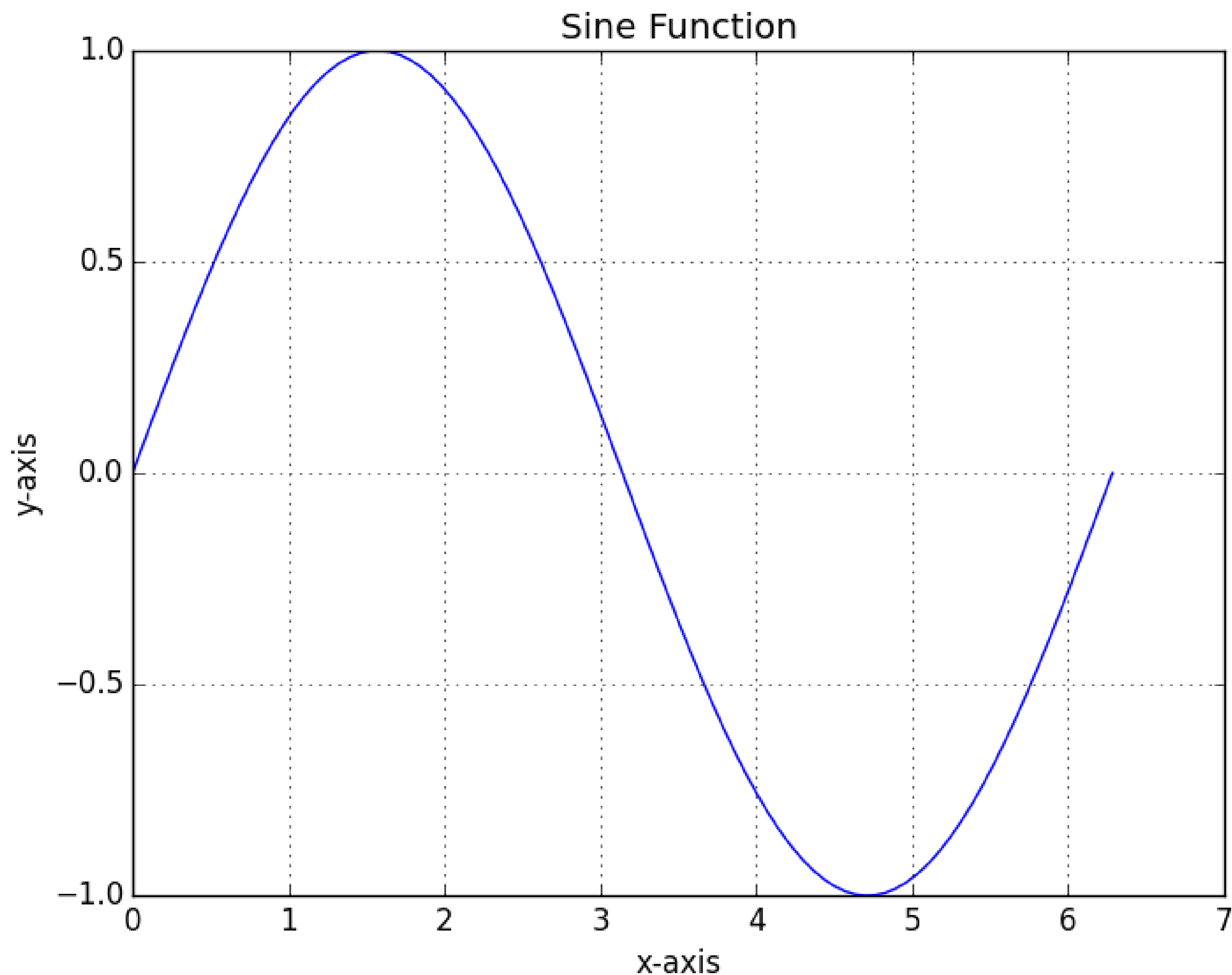
Abaixo temos o gráfico da função seno, gerado pelo programa *sine1.py*.



Como em toda biblioteca gráfica, podemos adicionar os nomes dos eixos, gradeado e título, com os métodos destacados em vermelho no código *sine2.py*. Os métodos *plt.xlabel()*, *plt.ylabel()* e *plt.title()* aceitam variáveis como argumentos, os strings, como no código abaixo.

```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine function
sine_func = np.sin(x)
# Creates plot
plt.plot(x, sine_func)
plt.xlabel('x-axis')           # Adds axis label
plt.ylabel('y-axis')           # Adds axis label
plt.title('Sine Function')     # Adds title
plt.grid(True)                 # Adds grid to the plot
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('sine2.png')
```

Abaixo temos o gráfico da função seno, gerado pelo programa *sine2.py*.



python™
www.python.org

O programa *trig1.py* gera o gráfico para as funções seno e cosseno. Para isto basta adicionarmos a variável *cos_func*, e atribuirmos a ela o resultado do método *np.cos(x)*.

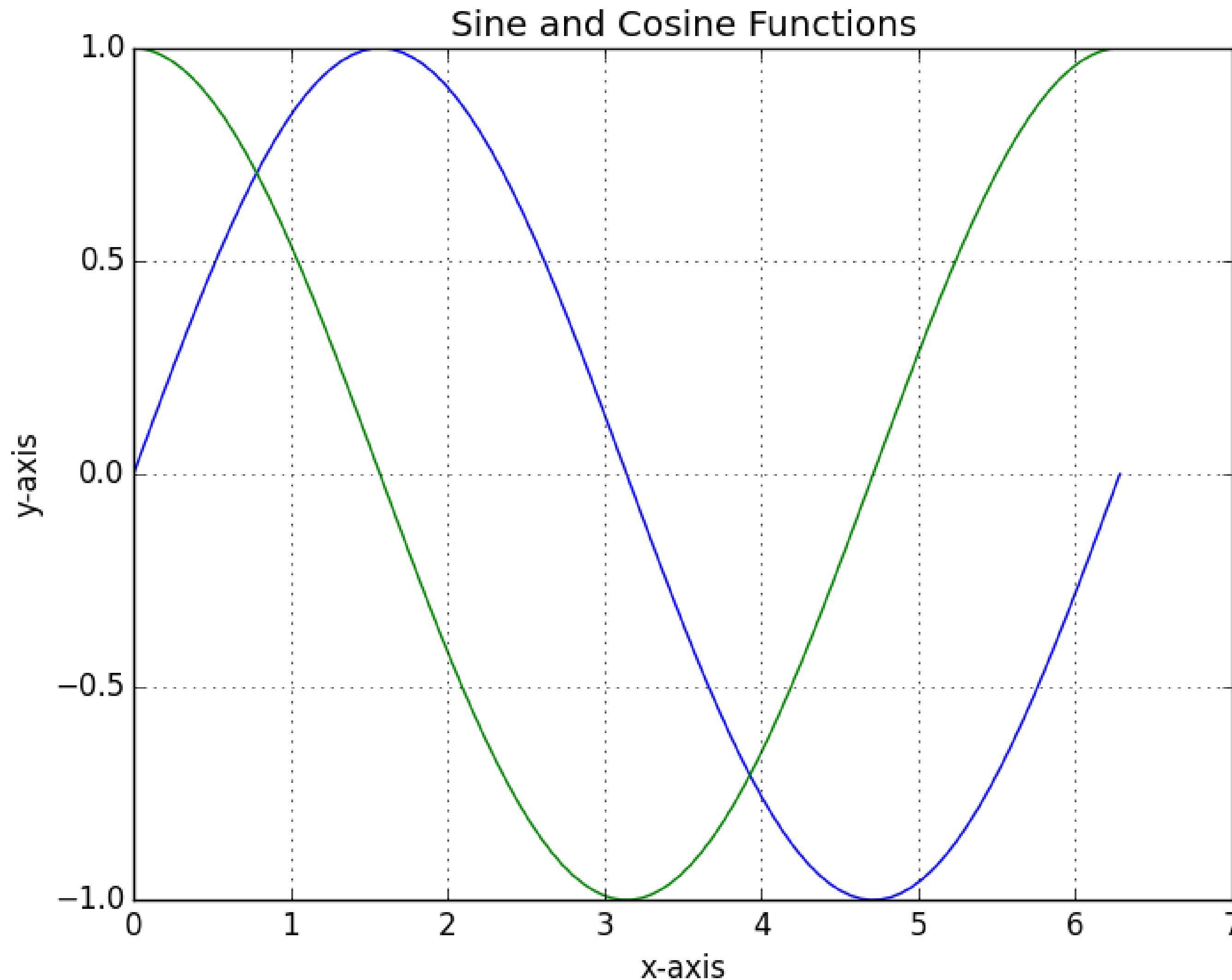
```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine and cosine functions
sine_func = np.sin(x)
cos_func = np.cos(x)
# Creates plots
plt.plot(x, sine_func)
plt.plot(x, cos_func)
plt.xlabel('x-axis')         # Adds axis label
plt.ylabel('y-axis')         # Adds axis label
plt.title('Sine and Cosine Functions')  # Adds title
plt.grid(True)               # Adds grid to the plot
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('trig1.png')
```

python™
www.python.org

Temos que adicionar o método *plt.plot(x,cos_func)* para o gráfico do cosseno, como indicado abaixo. Mudamos também o título do gráfico.

```
import matplotlib.pyplot as plt
import numpy as np
# Defines x range with linspace
x = np.linspace(0, 2*np.pi, 100)
# Defines sine and cosine functions
sine_func = np.sin(x)
cos_func = np.cos(x)
# Creates plots
plt.plot(x, sine_func)
plt.plot(x, cos_func)
plt.xlabel('x-axis')          # Adds axis label
plt.ylabel('y-axis')          # Adds axis label
plt.title('Sine and Cosine Functions')  # Adds title
plt.grid(True)                # Adds grid to the plot
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('trig1.png')
```

Abaixo temos o gráfico das funções seno cosseno, geradas pelo programa *trig1.py*. Veja que linha do cosseno é verde.



python™
www.python.org

Gráfico das funções seno e cosseno

Programa: *trig2.py*

Resumo
Programa para gerar gráficos das funções seno e cosseno, a partir das bibliotecas *Matplotlib* e *NumPy.* O usuário digita a faixa de valores para o eixo *x*, os rótulos dos eixos *x* e *y*, bem como o título do gráfico e o nome do arquivo de saída para o gráfico. Ambas funções trigonométricas são colocadas no mesmo gráfico. O programa gera um arquivo de saída no formato PNG.

python™
www.python.org

A principal novidade do código *trig2.py*, é que as informações são lidas a partir de entradas digitadas pelo usuário. Para isto precisamos da função *input()*, como indicado no trecho em vermelho abaixo. A leitura dos valores mínimo e máximo do eixo *x*, são convertidos para *float*. As informações são atribuídas às variáveis *x_min* e *x_max.* As outras informações são strings, e não devem ser convertidas.

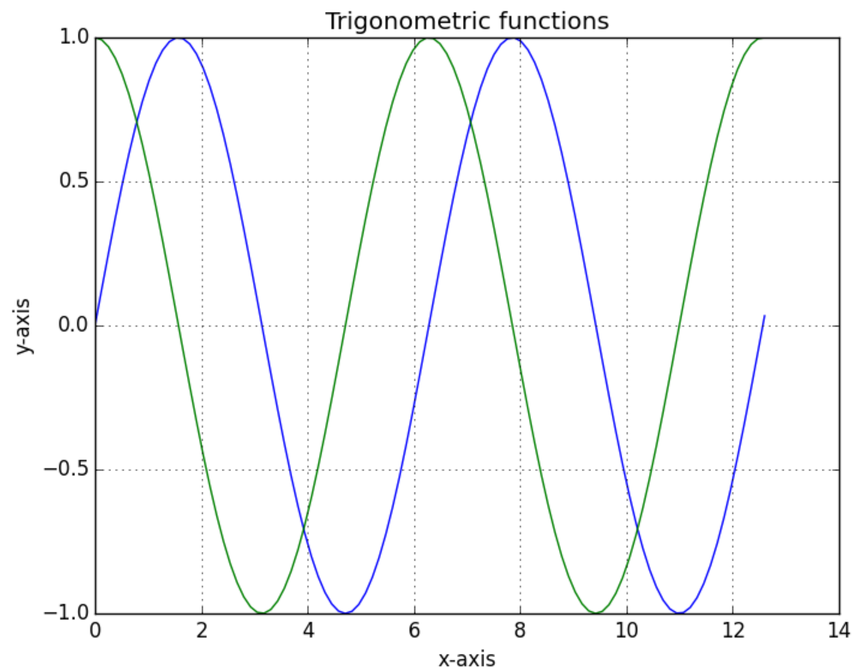
```
# Imports libraries
import matplotlib.pyplot as plt
import numpy as np

# Reads input information
x_min = float(input("Type minimum value for x-axis => "))
x_max = float(input("Type maximum value for x-axis => "))
x_label = input("Type x-axis label => ")
y_label = input("Type y-axis label => ")
plot_title = input("Type plot title => ")
output_file = input("Type output file name => ")
```

Os valores atribuídos às variáveis de entrada, são usados como argumentos para a chamada de funções para gerarmos o gráfico. As linhas de código, que usam os valores atribuídos às variáveis, estão indicadas em vermelho. A função *plt.grid(True)* não precisa de variável, ele tem como argumento o valor lógico *True*, que habilita o desenho de grades no gráfico.

```
# Defines x range with np.linspace()
x = np.linspace(x_min, x_max, 100)

# Defines sine and cosine functions
sine_func    = np.sin(x)
cos_func     = np.cos(x)

# Creates plots
plt.plot(x,sine_func)
plt.plot(x,cos_func)
plt.xlabel(x_label)       # Adds axis label
plt.ylabel(y_label)       # Adds axis label
plt.title(plot_title)     # Adds title
plt.grid(True)            # Adds grid to the plot

# Shows plot
plt.show()
# Saves plot
plt.savefig(output_file)
```

Abaixo temos os dados de entrada e o gráfico gerado.

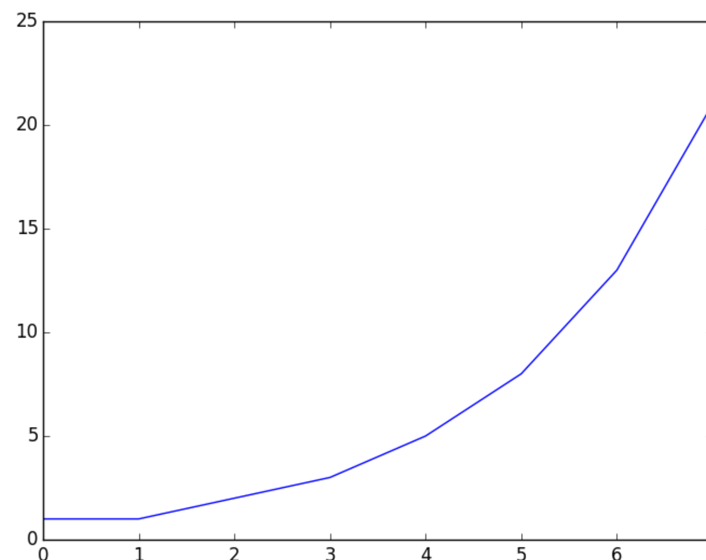
```
Type minimum value for x-axis => 0
Type maximum value for x-axis => 12.6
Type x-axis label => x-axis
Type y-axis label => y-axis
Type plot title => Trigonometric functions
Type output file name => trig2.png
```



Já vimos alguns programas para gráficos de funções trigonométricas. Veremos agora, com mais detalhes, diversos programas em Python para gerar gráficos, a partir de recursos da biblioteca *Matplotlib*. Abaixo temos um código simples, para exibição de pontos num gráfico bidimensional. As linhas de código estão em vermelho, os comentários em preto. A primeira linha importa a biblioteca *Matplotlib* como *plt*. Depois temos a linha com o *.plot()*, onde foram especificados 8 valores, que representam coordenadas no eixo y. A biblioteca *Matplotlib* usa valores implícitos para o eixo x, começando em zero até o valor inteiro N-1, onde N é o número de itens na lista. A função *.show()* mostra o gráfico na tela. Por último, a função *.savefig()*, salva o arquivo com o gráfico gerado. Veja, com 4 linhas de código, geramos o gráfico.

```
import matplotlib.pyplot as plt

# Generates a simple plot
plt.plot([1,1,2,3,5,8,13,21])

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("simple_plot1.png")
```

Abaixo temos o gráfico gerado.




```
import matplotlib.pyplot as plt

# Generates a simple plot
plt.plot([1,1,2,3,5,8,13,21])

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("simple_plot1.png")
```

python™
www.python.org

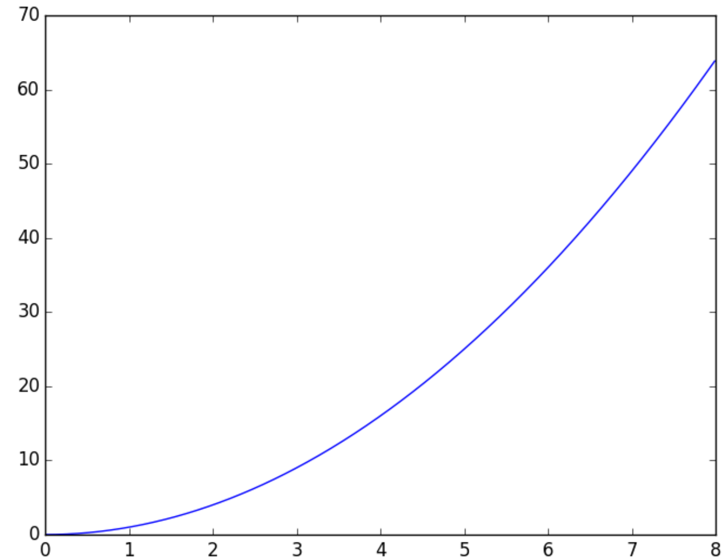
A função *range(i,j,k)* gera uma lista de inteiros, iniciando em *i* e finalizando em *j*, com um passo *k*. Usaremos uma função da biblioteca *NumPy*, chamada *np.arange(x,y,z)*, que gera uma sequência de números, não necessariamente inteiros. Esta sequência inicia em *x*, termina em *y-z* e tem um passo de *z*. Vejamos o código do programa *simple_plot2.py*, que gera uma parábola entre zero e 7.99. As linhas de código estão em vermelho e os comentários em preto.

```
# Program to illustrate the use of Matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis
y = x**2

# Generates plot
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("simple_plot2.png")
```

Inicialmente importamos a bibliotecas *Matplotlib* e *NumPy*. Em seguida chamamos a função *np.arange()* que gera uma sequência de números, entre 0 e 7.99, com passo 0.01. Depois geramos o eixo *y*, com *y* = *x***2. Com *plt.plot(x,y)* geramos os gráfico. O *plt.show()* mostra o gráfico na tela e *plt.savefig()* salva o arquivo com o gráfico. Efetivamente com 7 linhas de código geramos nosso gráfico.

```
# Program to illustrate the use of Matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis
y = x**2

# Generates plot
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("simple_plot2.png")
```

Abaixo temos o gráfico gerado.




```
# Program to illustrate the use of Matplotli
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis
y = x**2

# Generates plot
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("simple_plot2.png")
```

python™
www.python.org

Vejamos agora um programa com gráficos múltiplos. O programa *multiple_plots1.py* gera três curvas num mesmo gráfico para a mesma faixa de *x*.

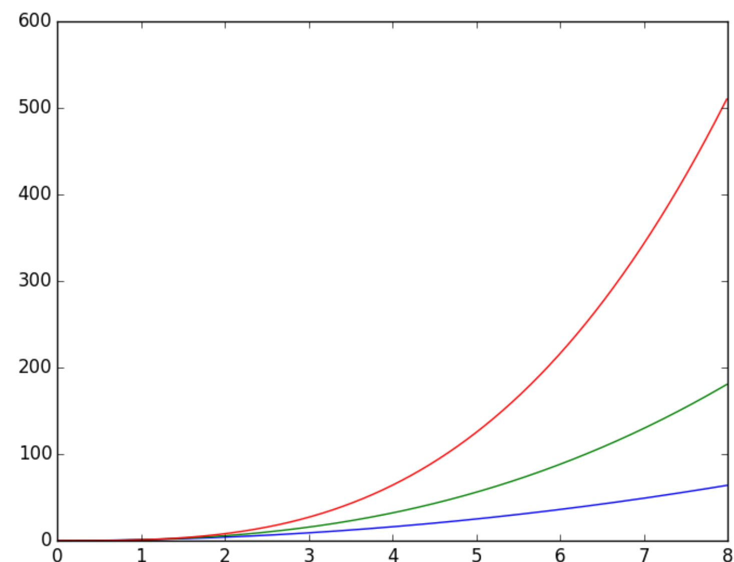
```
# Program to illustrate the use of Matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis and respective plot
y = x**2
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**2.5
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**3
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("multiple_plots1.png")
```

A principal novidade do código, é que geramos o eixo *y* três vezes e, em cada vez, chamamos o *plt.plot(x,y)*. Como mudamos o *y*, temos três gráficos distintos.

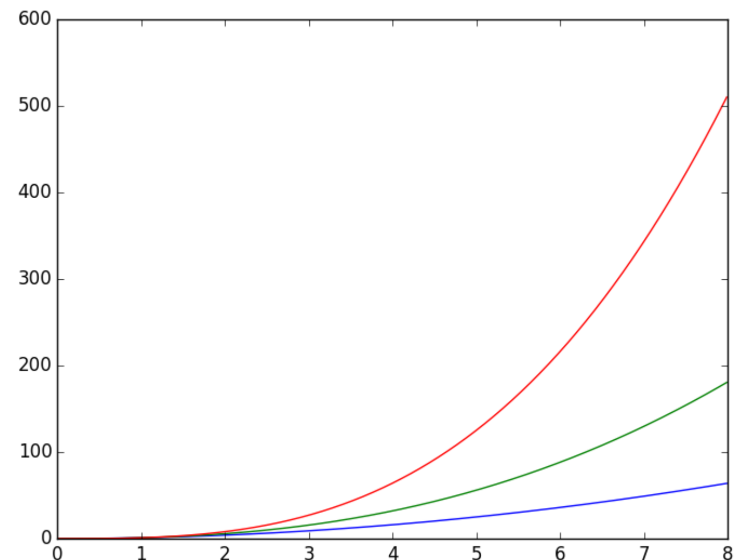
```
# Program to illustrate the use of Matplotlib
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis and respective plot
y = x**2
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**2.5
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**3
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("multiple_plots1.png")
```

Abaixo temos o gráfico gerado. Veja que a biblioteca *Matplotlib* gera as curvas em cores diferentes, a primeira curva em azul, depois verde e por último vermelha.

```
# Program to illustrate the use of Matplotli
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates y-axis and respective plot
y = x**2
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**2.5
plt.plot(x,y)

# Generates y-axis and respective plot
y = x**3
plt.plot(x,y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("multiple_plots1.png")
```



Podemos gerar as três curvas com um *plt.plot()* somente, como indicado na linha de código em vermelho abaixo. O programa *multiple_plots2.py* tem um número menor de linhas e gera o mesmo gráfico.




```
# Program to illustrate the use of Matplotli
import matplotlib.pyplot as plt
import numpy as np

# Generates x-axis
x = np.arange(0,8,0.01)

# Generates all plots in one command
plt.plot(x,x**2,x,x**2.5,x,x**3  )

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("multiple_plots2.png")
```

Vimos com o programa *trig2.py*, como incluir informações diversas, tais como, rótulos para os eixos, título do gráfico, além do gradeado de fundo do gráfico, como se fosse um papel milimetrado. Agora vamos colocar legendas no nosso gráfico. Para incluirmos tais legendas, adicionamos a palavra chave *label*, como argumento da função *plt.plot()*. Por exemplo, para adicionarmos a legenda “Function x**2”, temos que usar o comando:

```
plt.plot(x,x**2,label="Function x**2" )
```

Após a inclusão da palavra chave “label”, precisamos da função *plt.legend()*, que insere a legenda criada no gráfico. As opções de posicionamento da legenda, com a palavra chave *loc*, são as seguintes: *best, upper right, upper left, lower left, lower right, right, center left, center right, lower center, upper center, center*. Abaixo temos o comando, para posicionarmos a legenda à esquerda acima.

```
plt.legend(loc='upper left')
```

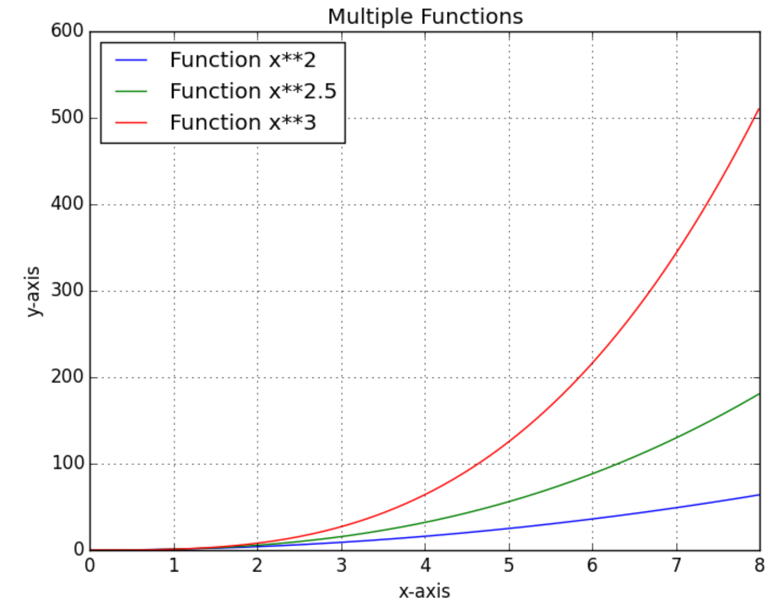
No próximo slide, temos o código completo para o programa *multiple_plot3.py* .

O programa *multiple_plot3.py* traz informações completas sobre o gráfico gerado.

```
# Program to illustrate the use of Matplotlib
import matplotlib.pyplot as plt
import numpy as np
# Generates x-axis
x = np.arange(0,8,0.01)

# Generates plots with legends
plt.plot(x,x**2,label="Function x**2" )
plt.plot(x,x**2.5,label="Function x**2.5" )
plt.plot(x,x**3,label="Function x**3" )

# Positioning the legends
plt.legend(loc='upper left')

plt.xlabel('x-axis')                    # Adds axis label
plt.ylabel('y-axis')                    # Adds axis label
plt.title('Multiple Functions')  # Adds title
plt.grid(True)                          # Adds grid to the plot

# Shows plot
plt.show()
# Saves plot on png file
plt.savefig("multiple_plots3.png")
```

O gráfico está mostrado abaixo.

```
# Program to illustrate the use of Matplotli
import matplotlib.pyplot as plt
import numpy as np
# Generates x-axis
x = np.arange(0,8,0.01)

# Generates plots with legends
plt.plot(x,x**2,label="Function x**2" )
plt.plot(x,x**2.5,label="Function x**2.5" )
plt.plot(x,x**3,label="Function x**3" )

# Positioning the legends
plt.legend(loc='upper left')
```

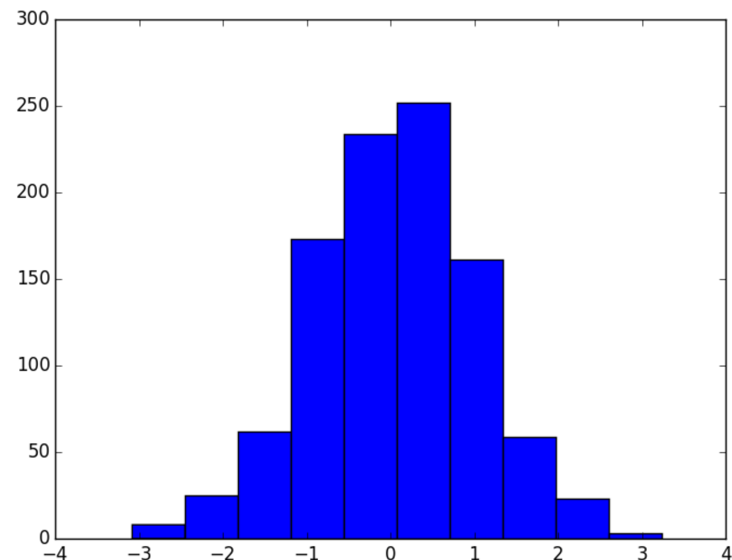



```
plt.xlabel('x-axis')                    # Adds axis label
plt.ylabel('y-axis')                    # Adds axis label
plt.title('Multiple Functions')  # Adds title
plt.grid(True)                          # Adds grid to the plot

# Shows plot
plt.show()
# Saves plot on png file
plt.savefig("multiple_plots3.png")
```

A biblioteca *Matplotlib* tem diversas opções de gráficos, como histograma. O programa *histogram_plot1.py* gera uma lista de números aleatórios e faz o gráfico da ocorrência destes números, dividido em faixas. Para gerar os números aleatórios, usamos a função *np.random.randn(1000),* que gera um *array* com 1000 números, que seguem uma distribuição gaussiana. Para mostrar o histograma, usamos *plt.hist.(y)*, que gera o histograma dividido em 10 faixas. O valor 10 é o padrão, podemos modificar o número de faixas, a partir da inclusão do número de faixas desejado, como argumento da função *plt.hist.()*, por exemplo: *plt.hist.(y,20)* gera um histograma com 20 divisões.

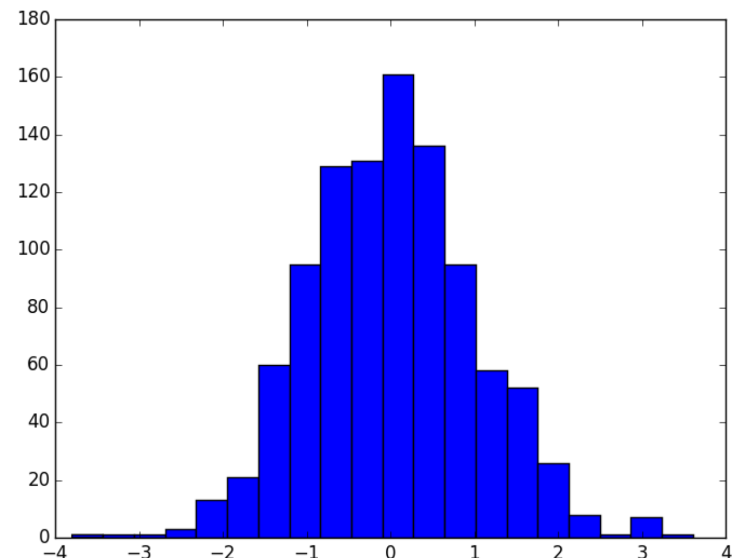
```
# Program to generate a histogram plot for a distribution of pseudo-random numbers

import matplotlib.pyplot as plt
import numpy as np

# Generates an array of pseudo-random numbers using np.random.randn()
y = np.random.randn(1000)

# Generates plot
plt.hist(y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("histogram_plot1.png")
```

Abaixo temos o histograma gerado, vemos que o eixo y indica a ocorrência da faixa indicada no eixo x.




```
# Program to generate a histogram plot for a
import matplotlib.pyplot as plt
import numpy as np

# Generates an array of pseudo-random number
y = np.random.randn(1000)

# Generates plot
plt.hist(y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("histogram_plot1.png")
```

O programa *histogram_plot2.py* gera um histograma com 20 divisões, como mostrada na figura abaixo. Só modificamos as linhas indicadas em vermelho.



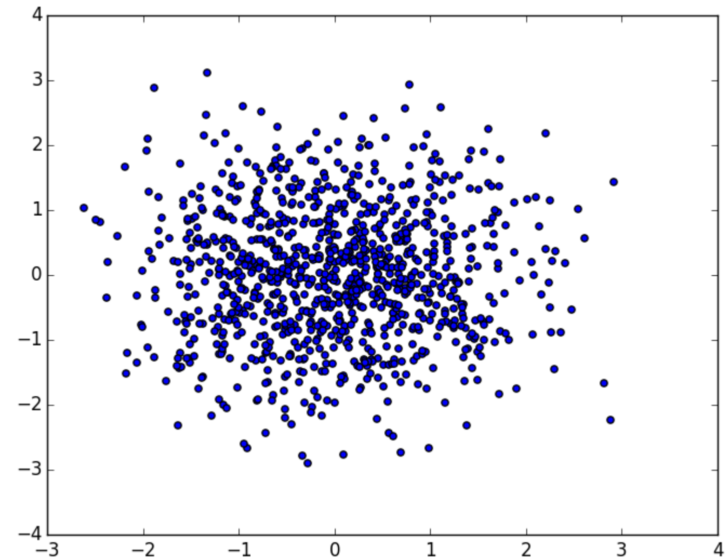

```
# Program to generate a histogram plot for a

import matplotlib.pyplot as plt
import numpy as np

# Generates an array of pseudo-random number
y = np.random.randn(1000)

# Generates plot
plt.hist(y, 20)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("histogram_plot2.png")
```

python™
www.python.org

Outro gráfico usado para estudo de distribuições, é o gráfico de dispersão. Na biblioteca *Matplotlib* temos a função *plt.scatter(),* que gera gráficos de dispersão. No programa *scatter_plot1.py,* temos a aplicação da função *np.random.randn(1000)* duas vezes, uma para gerar os números do eixo *x* e outra para gerar os números do eixo *y*. A função *plt.scatter(x,y)* faz o gráfico de distribuição dos números aleatórios. O código está mostrado abaixo.

```
# Program to generate a scatter plot for two distributions of pseudo-random numbers

import matplotlib.pyplot as plt
import numpy as np

# Generates two arrays of pseudo-random numbers using np.random.randn()
x = np.random.randn(1000)
y = np.random.randn(1000)

# Generates plot
plt.scatter(x, y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("scatter_plot1.png")
```

Abaixo temos o gráfico de espalhamento gerado.

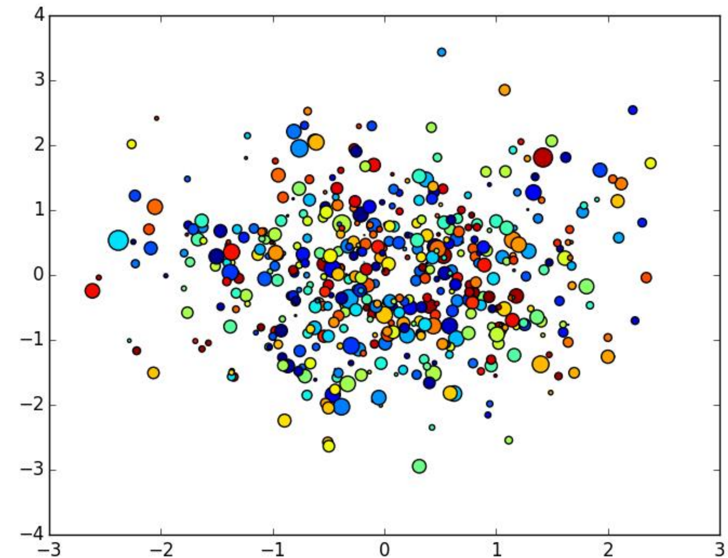



```
# Program to generate a scatter plot for two

import matplotlib.pyplot as plt
import numpy as np

# Generates two arrays of pseudo-random numb
x = np.random.randn(1000)
y = np.random.randn(1000)

# Generates plot
plt.scatter(x, y)

# Shows plot
plt.show()

# Saves plot on png file
plt.savefig("scatter_plot1.png")
```

Vamos modificar o código *scatter_plot1.py*, para que tenhamos cores nos pontos do gráfico. Além disso, vamos variar os tamanhos dos pontos. As cores e os tamanhos dos pontos irão variar de forma aleatória. Para gerarmos cores, usarmos a palavra chave *c* como argumento da função *plt.plot()*. No caso do tamanho do ponto do gráfico, usamos a palavra chave *s*. Assim, criamos distribuições aleatórias para variáveis *size* e *colors*. Para a variável *size*, temos que multiplicar por 50, visto que é expresso em pixels. O código está mostrado abaixo.

```
# Program to generate a scatter plot for two distributions of pseudo-random numbers,
using also random number
# for size and colors
import matplotlib.pyplot as plt
import numpy as np
# Generates four arrays of pseudo-random numbers using np.random.randn()
x = np.random.randn(1000)
y = np.random.randn(1000)
size = 50*np.random.randn(1000)
colors = np.random.rand(1000)
# Generates plot
plt.scatter(x, y, s = size, c = colors)

# Shows plot
plt.show()
# Saves plot on png file
plt.savefig("scatter_plot2.png")
```

O gráfico gerado está mostrado abaixo.




```
# Program to generate a scatter plot for two                               s,
using also random number
# for size and colors
import matplotlib.pyplot as plt
import numpy as np
# Generates four arrays of pseudo-random num
x = np.random.randn(1000)
y = np.random.randn(1000)
size = 50*np.random.randn(1000)
colors = np.random.rand(1000)
# Generates plot
plt.scatter(x, y, s = size, c = colors)

# Shows plot
plt.show()
# Saves plot on png file
plt.savefig("scatter_plot2.png")
```

Do ponto de vista de aplicação científica, seria interessante a possibilidade de lermos os pontos dos *arrays* *x* e *y* a partir de um arquivo externo. Você provavelmente já fez este processo, por exemplo, quando abriu uma planilha do *Excel*. O programa *Excel* usa como um dos formatos para suas planilhas, o formato CSV (*Comma Separated Values*). Arquivo no formato CSV tem os valores separados por vírgulas, como mostrado no trecho abaixo.

```
Name,MolDock Score,Rerank Score,RMSD,Interaction,Internal,HBond,LE1,LE3,Docking Score,DisplacedWater
[291]SCF_501 [A],-135.265,-108.212,0.550558,-136.511,16.8594,-3.70663,-5.63606,-4.50883,-135.927,-15.6138
[278]SCF_501 [A],-135.279,-108.201,0.552119,-136.537,16.8903,-3.69755,-5.63661,-4.5084,-135.93,-15.632
[261]SCF_501 [A],-135.287,-108.197,0.553228,-136.515,16.8589,-3.69596,-5.63694,-4.50822,-135.939,-15.6301
[258]SCF_501 [A],-135.29,-108.195,0.551817,-136.507,16.8532,-3.69173,-5.63708,-4.50813,-135.942,-15.6365
[267]SCF_501 [A],-135.289,-108.18,0.553158,-136.595,16.9356,-3.68408,-5.63702,-4.5075,-135.935,-15.6291
[299]SCF_501 [A],-135.282,-108.164,0.554487,-136.586,16.9128,-3.67745,-5.63676,-4.50681,-135.925,-15.609
[256]SCF_501 [A],-135.297,-108.161,0.554416,-136.553,16.8902,-3.68148,-5.63736,-4.5067,-135.943,-15.6342
[252]SCF_501 [A],-135.295,-108.159,0.553906,-136.553,16.8732,-3.68676,-5.63729,-4.50663,-135.947,-15.6148
[257]SCF_501 [A],-135.291,-108.158,0.553521,-136.558,16.8822,-3.68676,-5.63712,-4.50657,-135.943,-15.6151
[265]SCF_501 [A],-135.291,-108.153,0.553381,-136.626,16.9517,-3.68131,-5.63712,-4.50638,-135.937,-15.6163
[272]SCF_501 [A],-135.283,-108.153,0.553865,-136.602,16.9356,-3.69094,-5.63678,-4.50639,-135.934,-15.6167
```

A função *np.genfromtxt('data1.csv', delimiter=",", skip_header = 1)* lê o arquivo *data1.csv* e considera as colunas de dados estão separadas por vírgulas, opção indicada por *delimiter=","*. Além disso, é desconsiderada a primeira linha do arquivo CSV, com a opção *skip_header = 1,* que pula a primeira linha. Caso quiséssemos pular duas linhas, usaríamos *skip_header = 2,* e assim sucessivamente. Abaixo temos a indicação dos principais argumentos da função *np.genfromtxt().*

```
np.genfromtxt('data1.csv', delimiter=",", skip_header = 1)
```

Número de linhas que serão puladas a partir da primeira, no caso pula-se a primeira linha.

Indica o símbolo usado para separar as colunas, no caso são vírgulas ",".

Nome do arquivo CSV

Abaixo temos os dados do arquivo *data1.csv*. Veja que *com skip_header* = *1*, pulamos a primeira linha, que tem os nomes dos eixos.

```
np.genfromtxt('data1.csv', delimiter=",", skip_header = 1)
```

```
x,y
0.1,1.25
0.2,1.5
0.3,4.0
0.4,5.1
0.5,6.3
0.6,5.0
0.7,8.7
0.8,9.0
0.9,11.0
1.0,14.0
1.1,13.75
1.2,15.0
1.3,18.0
1.4,17.5
1.5,19.0
1.6,17.0
1.7,21.0
1.8,22.5
1.9,26.0
2.0,25.1
```

Para a leitura das colunas do arquivo CSV, usamos o comando *x = my_csv[:,0],* que lê toda a primeira coluna (coluna zero). Abaixo temos um arquivo CSV, cada número no arquivo é acessado de forma matricial. Assim, considerando que começamos no número zero, o número na caixa vermelha é o *my_csv[5,0]*. O número da caixa azul é o *my_csv[11,1]*.

```
0.1,1.25
0.2,1.5
0.3,4.0
0.4,5.1
0.5,6.3
0.6,5.0
0.7,8.7
0.8,9.0
0.9,11.0
1.0,14.0
1.1,13.75
1.2,15.0
1.3,18.0
1.4,17.5
1.5,19.0
1.6,17.0
1.7,21.0
1.8,22.5
1.9,26.0
2.0,25.1
```

Linha 5, coluna 0

Linha 11, coluna 1

Para varremos toda coluna zero, temos que manter o número da coluna zero, e varrer todas as linhas, ou seja, poderíamos usar *my_csv[0:19,0]*. A indicação *0:19* representa que tomamos da linha 0 até a linha 19. O problema é que este comando funcionaria só para arquivo com 20 linhas. Para tonarmos de uso geral, para qualquer número de linha, usamos a faixa, com :, ou seja, *my_csv[:,0]*.

```
0.1,1.25
0.2,1.5
0.3,4.0
0.4,5.1
0.5,6.3
0.6,5.0
0.7,8.7
0.8,9.0
0.9,11.0
1.0,14.0
1.1,13.75
1.2,15.0
1.3,18.0
1.4,17.5
1.5,19.0
1.6,17.0
1.7,21.0
1.8,22.5
1.9,26.0
2.0,25.1
```

Linha 5, coluna 0

Linha 11, coluna 1

python™
www.python.org

O código está mostrado abaixo. O bloco em vermelho destaca a parte da leitura do arquivo CSV, *x = my_csv[:,0]* lê toda a primeira coluna (coluna zero).

```
import numpy as np
import matplotlib.pyplot as plt
# Reads CSV file
my_csv = np.genfromtxt ('data1.csv', delimiter=",", skip_header = 1)
# Gets each column from CSV file
x = my_csv[:,0]
y = my_csv[:,1]
# Generates plot
plt.scatter(x,y)
# Least-squares polynomial fitting
z = np.polyfit(x,y, 1)
p = np.poly1d(z)
print("Best fit polinomial equation: ",p)
# Generates plot
plt.plot(x, p(x), '-')
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('scatter_plot3.png')
```

python™
www.python.org

A função *polyfit(x,y,1)* gera uma reta aproximada para os pontos (x,y). O número “1” indica uma reta (grau 1). Podemos gerar modelos para outros polinômios (graus 2,...).

```
import numpy as np
import matplotlib.pyplot as plt
# Reads CSV file
my_csv = np.genfromtxt ('data1.csv', delimiter=",", skip_header = 1)
# Gets each column from CSV file
x = my_csv[:,0]
y = my_csv[:,1]
# Generates plot
plt.scatter(x,y)
# Least-squares polynomial fitting
z = np.polyfit(x,y, 1)
p = np.poly1d(z)
print("Best fit polynomial equation: ",p)
# Generates plot
plt.plot(x, p(x), '-')
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('scatter_plot3.png')
```

python™
www.python.org

A função *poly1d(x)* gera a função matemática determinada pela função *polyfit(x,y,1)*, que foi atribuída à variável *p*.

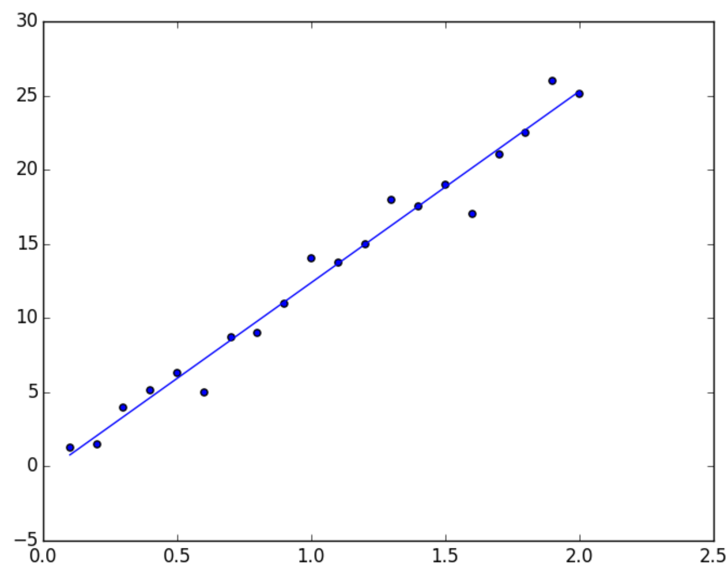
```
import numpy as np
import matplotlib.pyplot as plt
# Reads CSV file
my_csv = np.genfromtxt ('data1.csv', delimiter=",", skip_header = 1)
# Gets each column from CSV file
x = my_csv[:,0]
y = my_csv[:,1]
# Generates plot
plt.scatter(x,y)
# Least-squares polynomial fitting
z = np.polyfit(x,y, 1)
p = np.poly1d(z)
print("Best fit polynomial equation: ",p)
# Generates plot
plt.plot(x, p(x), '-')
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('scatter_plot3.png')
```

Ao usarmos *p(x)*, temos um *array* onde para cada elemento do *array x* foi calculado o valor *p(x)*. No caso os valores de *p(x)* serão usados como coordenadas do eixo *y*.

```
import numpy as np
import matplotlib.pyplot as plt
# Reads CSV file
my_csv = np.genfromtxt ('data1.csv', delimiter=",", skip_header = 1)
# Gets each column from CSV file
x = my_csv[:,0]
y = my_csv[:,1]
# Generates plot
plt.scatter(x,y)
# Least-squares polynomial fitting
z = np.polyfit(x,y, 1)
p = np.poly1d(z)
print("Best fit polynomial equation: ",p)
# Generates plot
plt.plot(x, p(x), '-')
# Shows plot
plt.show()
# Saves plot on png file
plt.savefig('scatter_plot3.png')
```

Abaixo temos informações sobre a reta obtida pelo programa e o gráfico gerado.

```
Best fit polynomial equation:
2.047 x - 0.2116
```



Scatter Plot (Versão 4)

Programa: *scatter_plot4.py*

Resumo
Programa para gerar o gráfico de espalhamento para um arquivo CSV cujo o nome é dado pelo usuário. O grau do polinômio também é fornecido pelo usuário. Teste o programa para o arquivo data2.csv.

-BRESSERT, Eli. SciPy and NumPy. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.

-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.

-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.

-KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python.** 2ed. Nova York: Cambridge University Press, 2010. 422 p.

-LANDAU, Rubin H. **A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90**. Princeton: Princeton University Press, 2005. 481p.

-LANDAU, Rubin H., PÁEZ, Manuel José, BORDEIANU, Cristian C. **A Survey of Computational Physics. Introductory Computational Physics.** Princeton: Princeton University Press, 2008. 658 p.

-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.

-MODEL, Mitchell L. **Bioinformatics Programming Using Python**. Sebastopol: O'Reilly Media, Inc., 2011. 1584 p.

-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.

Última atualização: 12 de junho de 2019.