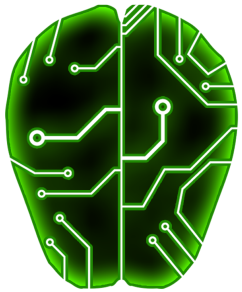
Curso de Python: Parte 2

Métodos Numéricos



COMPUTER SOCIETY

IFB

Guilherme Paiva

Instituto Federal de Brasília, Campus Taguatinga

Sumário

Módulos

- Aprendemos a importar módulos, mas como veremos quais estão instalados?

```
>>>help("modules")
```

Instalação

- Windows:

```
python -m pip install NomeDoPacote
python -m pip install --upgrade NomeDoPacote
```

- Linux:

```
sudo pip3 install NomeDoPacote
```

Numpy

- É um módulo do Python para computação científica
- Suporta Arrays e Matrizes multidimensionais
- Funções de algebra linear, geração de números pseudo-aleatórios

Numpy Arrays

- Partindo de Listas ou Tuplas

```
import numpy as np
a = np.array([1,2,3,4])
print(type(a))
print(a.shape)
b = np.array([[1,2,3],[4,5,6]], dtype = 'float')
```

Numpy Arrays

- Arrays com tamanhos pré-definidos

```
from numpy import pi
a = np.zeros((2,2)) #duas dimensões
b = np.ones(5)  #uma dimensão
c = np.arange(1,5,.5) #inicial, final, step
d = np.linspace(0,2*pi, 10) #inicial, final, n elementos
f = np.sin(d)
```

Operações com Arrays

- Operadores aritméticos atuam em cada elemento individualmente

```
import numpy as np
a = np.arange(9).reshape(3,3)
b = np.arange(4)
c = np.arange(10,50,10)
print(c-b)
print(b**2)
print(a*b)
print(a@c)
```

Funções Gerais

- mmc = np.lcm(x1,x2)
- mdc = np.gcd(x1,x2)
- np.log(x), np.log10(x), np.log2(x)
- np.cos(x), np.atanh(x),. . .
- np.mean(x), np.sum('axis')
- np.sign(x)

Matrizes

```
import numpy as np
mat_rand = np.random.rand(4,5)
mat_id = np.eye(9)
det_mat = np.linalg.det(mat_rand)
```

Scipy

- É um pacote científico do Python que utiliza como base o Numpy. Possui vários módulos, são eles:
  - Estatística
  - Otimização
  - Integração Numérica
  - Álgebra Linear
  - Processamento de sinais e imagens
  - Solução de equações diferenciais
  - Polinômios

Otimização

- Dentro do pacote de Otimização, podemos determinar:
  - Pontos de máximos e mínimos de funções
  - Raízes de equações

Raízes de equações

- Utilizamos algumas funções do módulo optimize para determinar raizes de equações:
  - bisect - Método da bissecção
  - fixed_point - Método do ponto fixo
  - newton - Método de Newton (deve-se fornecer a função derivada)

Bissecção

```
from scipy import optimize
def f(x):
    return x**2 - 1
raiz = optimize.bisect(f,-4,0)
print(raiz)
```

Ponto Fixo

```
from scipy import optimize
def f(x):
    return x**2 - 1
raiz = optimize.fixed_point(f,3)
print(raiz)
```

Newton

```
from scipy import optimize
def f(x):
    return x**3 + 8
def der_f(x):
    return 3 * x**2
raiz = optimize.newton(f,20.0,der_f)
print(raiz)
```

Matplotlib

- É uma biblioteca do Python utilizada para plotar gráficos em alta qualidade
- Ferramenta essencial para visualização de dados
- É open source, não é necessário adquirir licença para o uso
- Integrado ao LaTeX

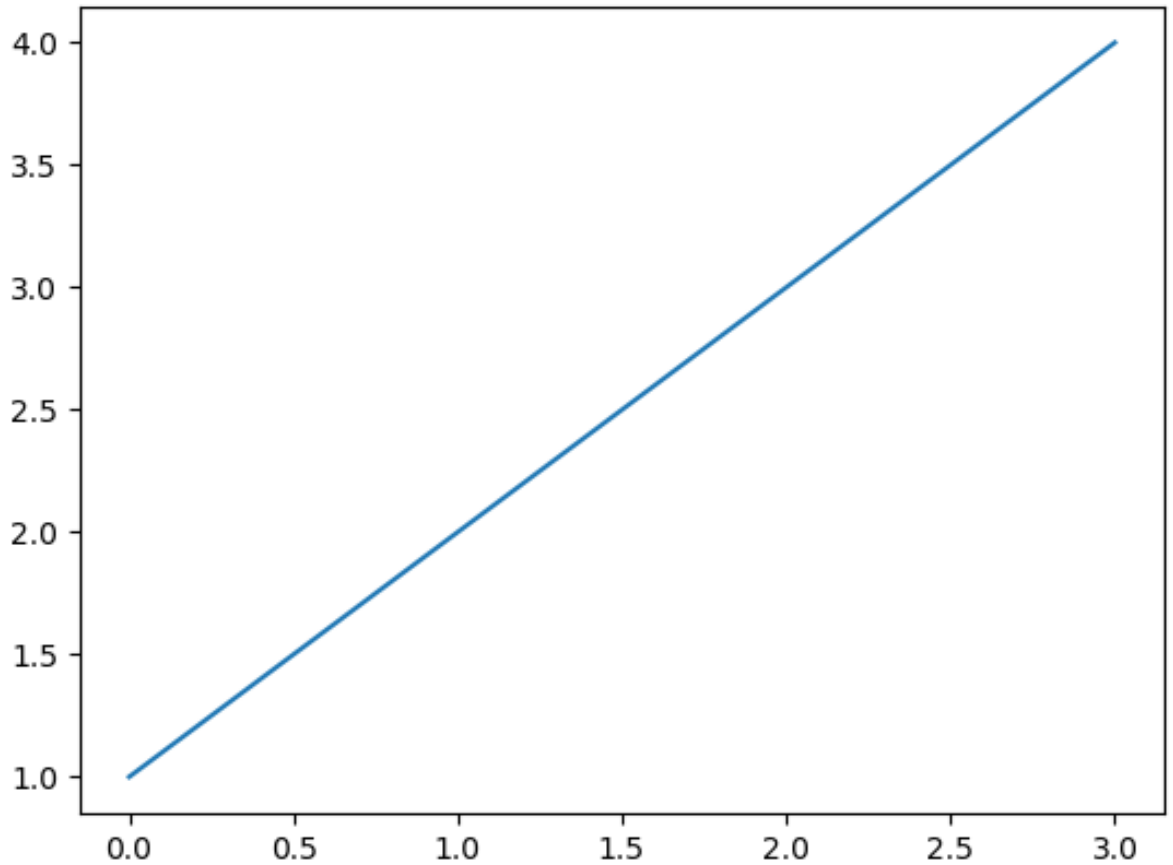
Básico de funções

- Domínio e imagem

Primeiro Gráfico

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.show()
```

Primeiro Gráfico



Primeiro Gráfico

- Como funciona?
  - **plt.plot()** recebe o conjunto de Domínio e Imagem
  - Caso o Domínio seja omitido, o pyplot gera automaticamente um conjunto de X para seu conjunto Y
  - Mas por que X vai até 3 sendo que Y vai até 4?
  - **plt.show()** mostra o gráfico em forma de imagem.

Adicionando Informações

- Temos números e uma reta, mas sem significados.
- Podemos preencher nosso gráfico de informações, para uma melhor visualização e entendimento.

Título

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('Título do eixo Y')
plt.xlabel('Título do eixo X')
plt.title('Título do Gráfico')
plt.show()
```

Gráfico com Título

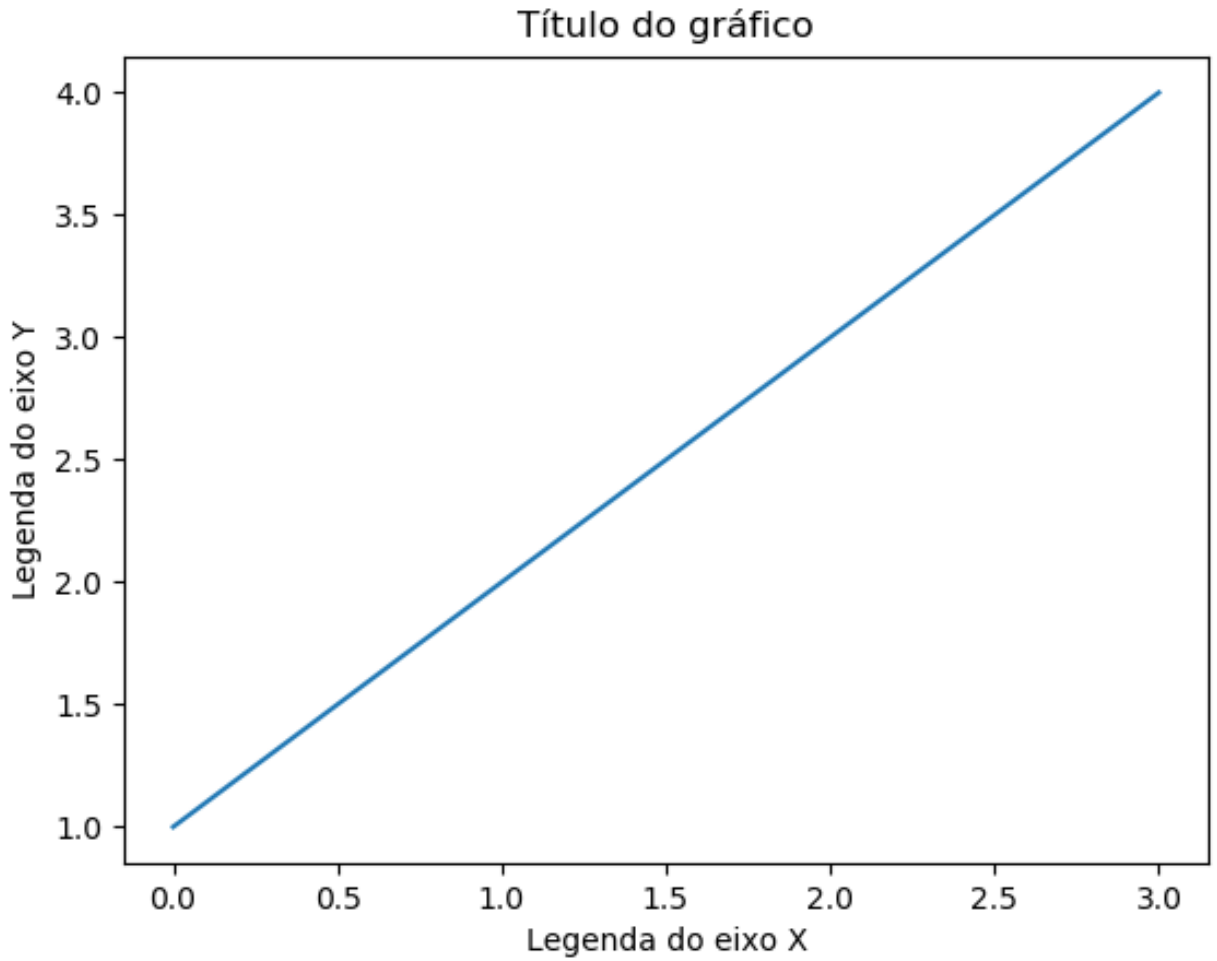


Gráfico com Título

- Como funciona?
  - **plt._label** adiciona um título ao eixo desejado
  - **plt.title** adiciona um título ao gráfico

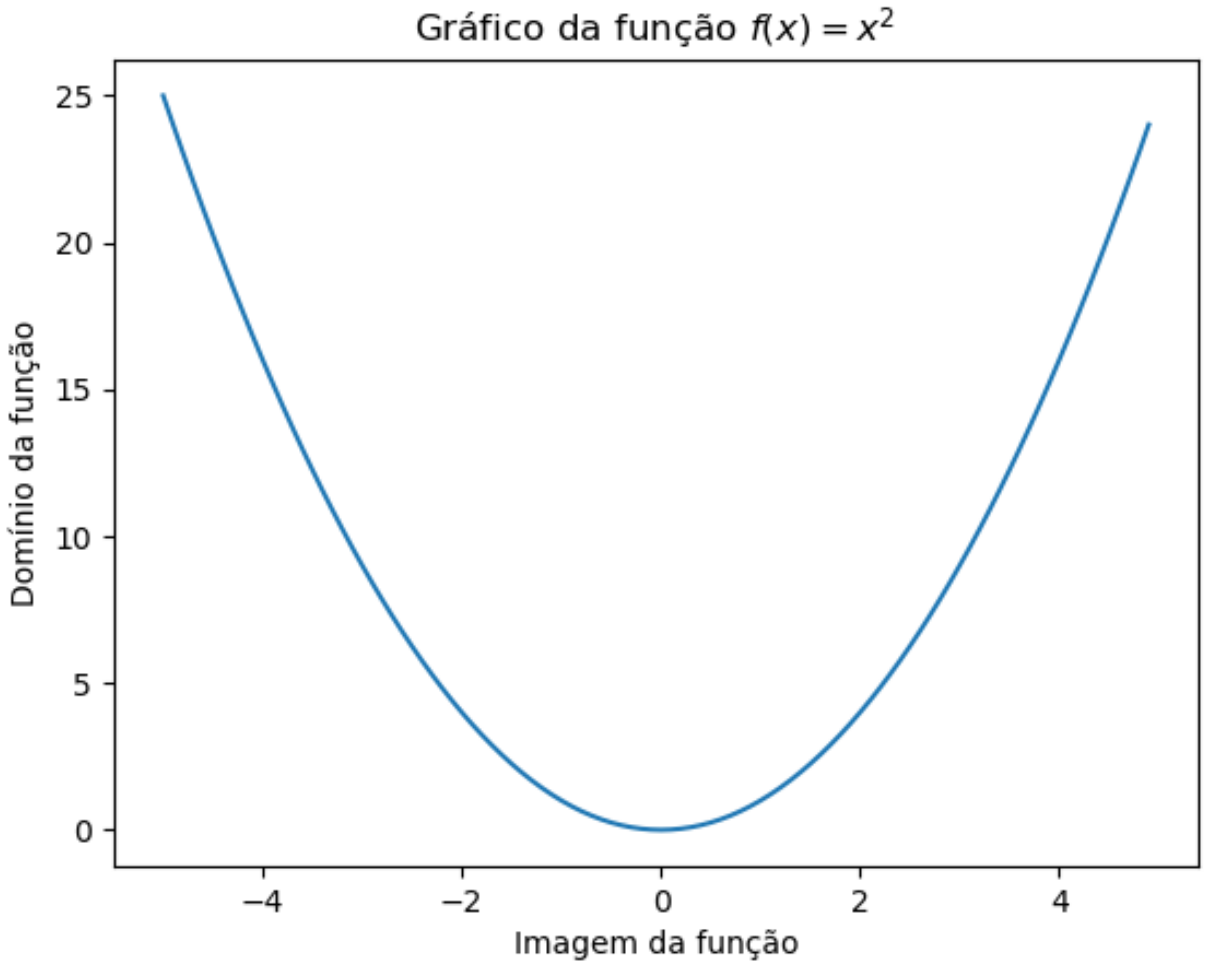
Domínio e imagem

- Podemos inserir os domínios e imagens que quisermos, desde que eles tenham a mesma dimensão
- Vamos usar nosso conhecimento já adquirido para gerar funções interessantes

Domínio e Imagem

```
import matplotlib.pyplot as plt
x = np.arange(-5,5,.1)
y = np.array([a**2 for a in x])
plt.plot(x,y)
plt.ylabel('Domínio da função')
plt.xlabel('Imagem da função')
plt.title('Gráfico da função '+r'$f(x) = x^2$')
plt.show()
```

Domínio e Imagem



Domínio e Imagem

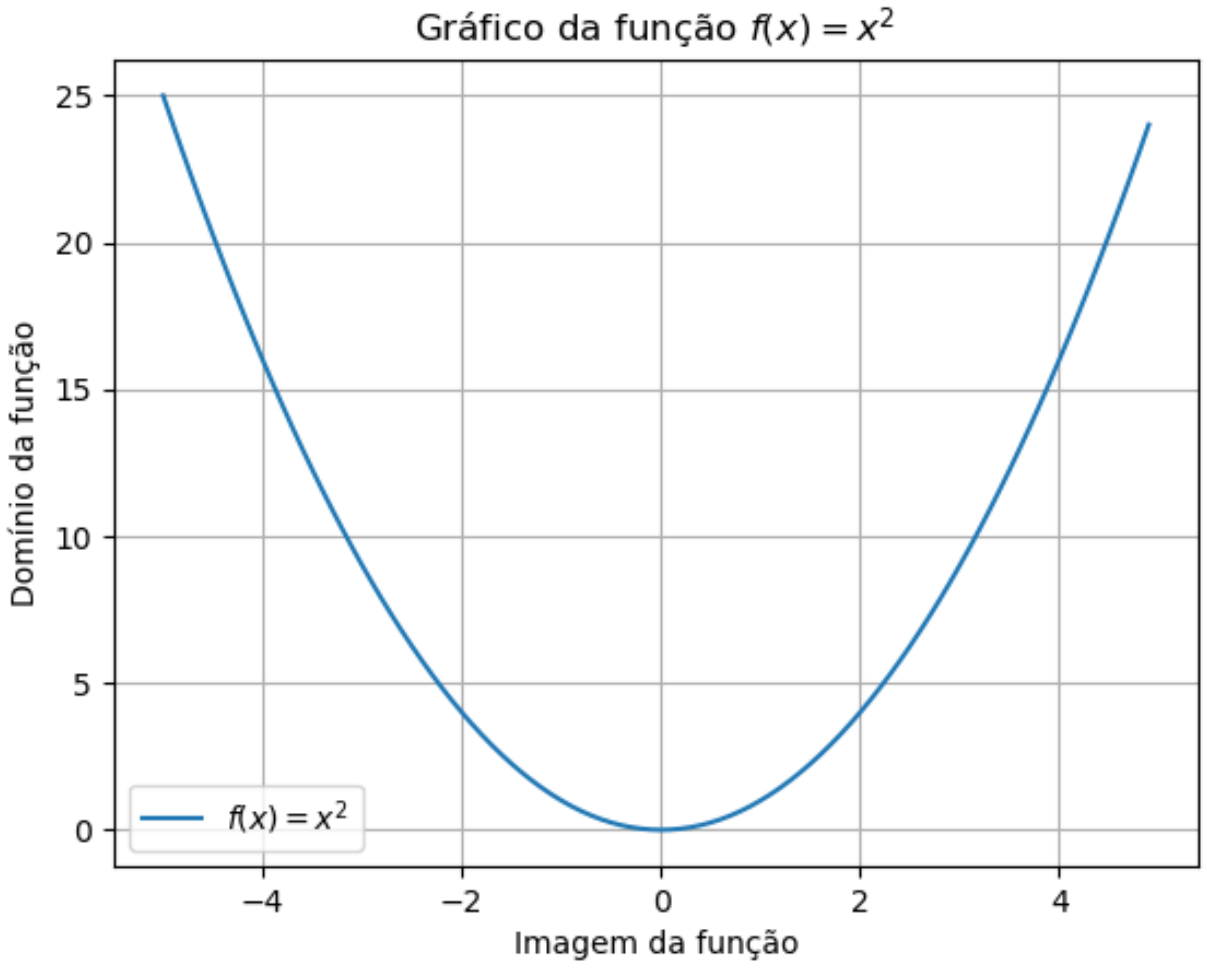
- Como funciona?
  - Utilizamos o numpy para gerar um conjunto domínio.
  - Geramos a imagem a partir do domínio e fizemos com que o programa plotasse nosso domínio e imagem.
  - Modo matemático no título.

Melhorando a visualização

```
import matplotlib.pyplot as plt

x = np.arange(-5,5,.1)
y = np.array([a**2 for a in x])
plt.plot(x,y, label=r'$f(x) = x^2$')
plt.ylabel('Domínio da função')
plt.xlabel('Imagem da função')
plt.title('Gráfico da função '+r'$f(x) = x^2$')
plt.legend(loc= 'best')
plt.grid(True)
plt.show()
```

Melhorando a visualização



Melhorando a visualização

- Como funciona?
  - Na função plot, especificamos uma legenda para a curva em questão com label.
  - Ao invocar o método grid com o parâmetro True, definimos que queremos exibir uma grade.

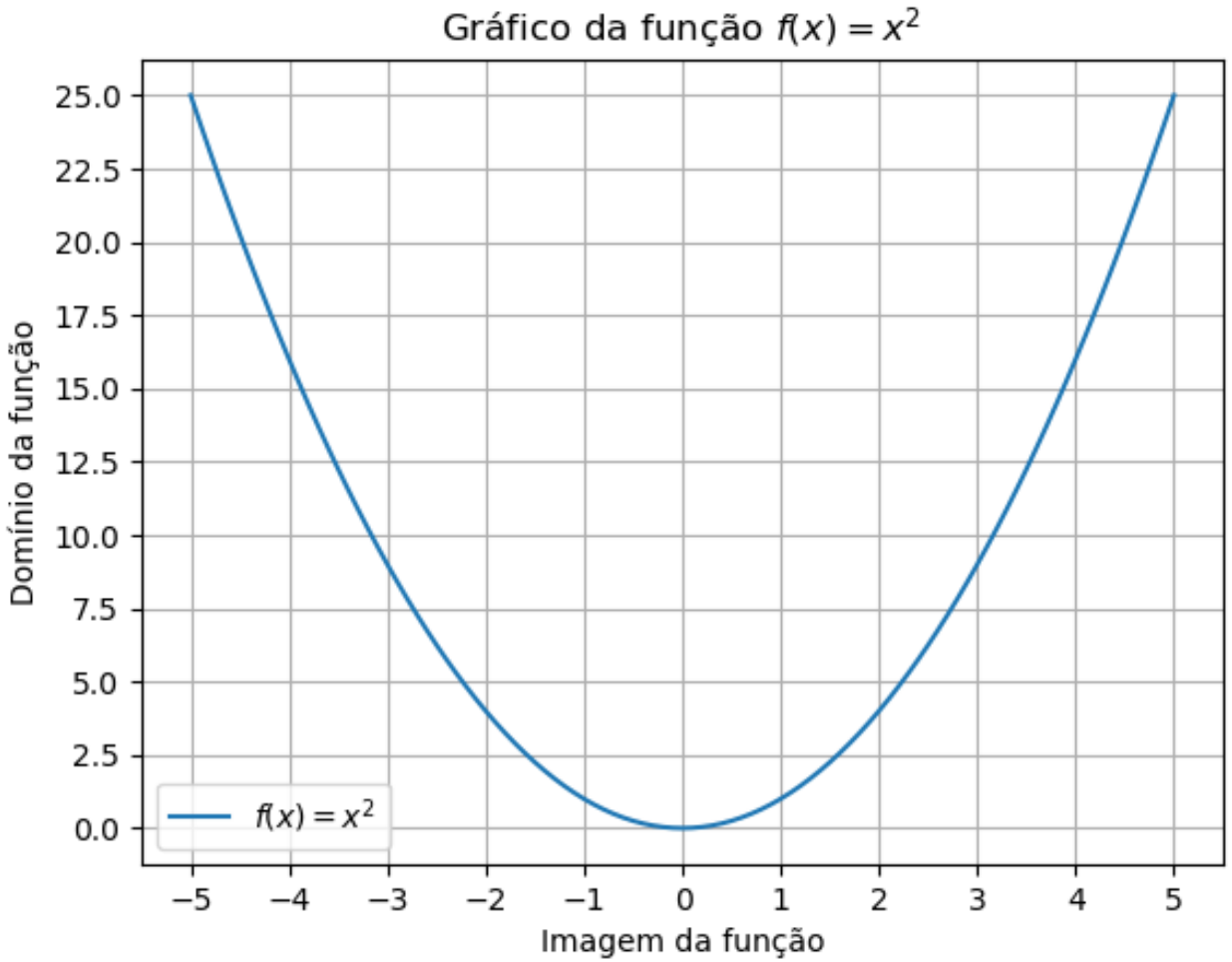
Marcadores

- E se nossa grid não estiver precisa o suficiente?
- Podemos usar marcadores para definir as distâncias das linhas de grid.

Marcadores

```
import matplotlib.pyplot as plt
x = np.arange(-5,5.1,.1)
y = np.array([a**2 for a in x])
plt.plot(x,y, label=r'$f(x) = x^2$')
plt.ylabel('Domínio da função')
plt.xlabel('Imagem da função')
plt.title('Gráfico da função '+r'$f(x) = x^2$')
plt.legend(loc= 'best')
plt.grid(True)
plt.xticks(np.arange(-5,6,1))
plt.yticks(np.arange(0,27.5,2.5))
plt.show()
```

Domínio e Imagem



Melhorando a visualização

- Como funciona?
  - A função plt._ticks recebe uma lista, gerada pelo numpy, com as coordenadas de onde ele deve gerar os marcadores
  - Note que também alteramos os ranges dos eixos pois o intervalo é aberto no final.

Personalizando as curvas

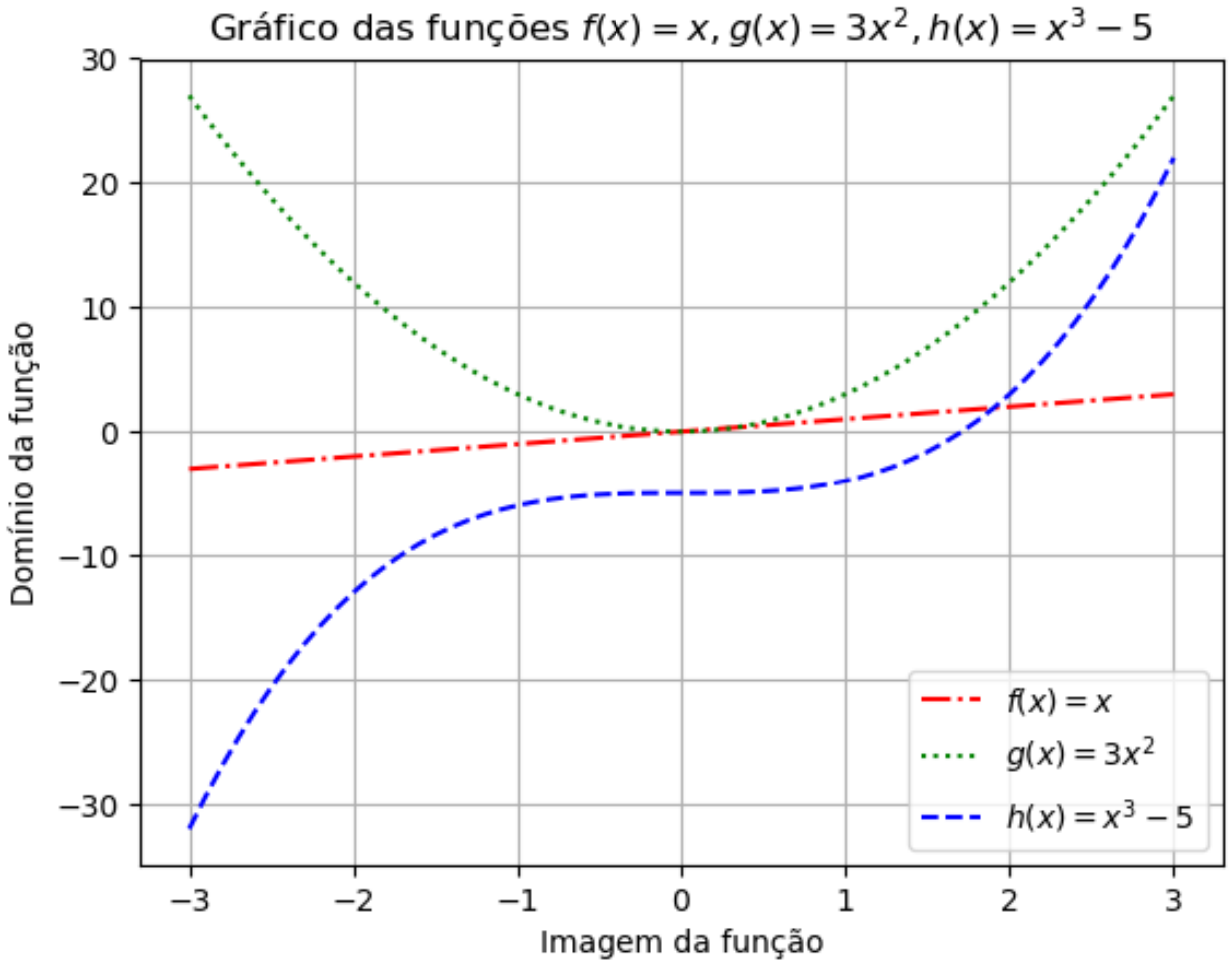
- É possível personalizar as curvas plotadas, escolhendo as cores e o estilo (tracejado, pontos, diamantes, etc).
- Na especificação da função plot(), podemos especificar a cor e o estilo
- Exemplo:

```
plt.plot(x,y,label=r"$f(x) = x^2$",color='red',linestyle='-.')
```

Modificando o estilo de curvas

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(-5,5,.1)
y1 = np.array([a for a in x])
y2 = np.array([3*a**2 for a in x])
y3 = np.array([a**3-5 for a in x])
plt.plot(x,y1,label=r"$f(x) = x$",color='red',linestyle='-.')
plt.plot(x,y2,label=r"$g(x) = 3x^2$",color='green',linestyle=':')
plt.plot(x,y3,label=r"$h(x) = x^3-5$",color='blue',linestyle='--')
plt.ylabel('Domínio da função')
plt.xlabel('Imagem da função')
plt.title('Gráfico das funções '+r'$f(x) = x, g(x) = 3x^2, h(x) = x^3-5$')
plt.legend(loc= 'best')
plt.grid(True)
plt.show()
```

Modificando o estilo de curvas



Múltiplos gráficos em uma única imagem

- Também é possível gerar diferentes gráficos em uma única imagem.
- Na definição dos eixos podemos ter o seguinte:

```
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)
```

- O que significam esses números?
  - Número de linhas.
  - Número de colunas.
  - Índice da célula.

Múltiplos gráficos em uma única imagem

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax2 = fig.add_subplot(212)
x = np.arange(0,10.5,0.5)
y1 = np.array([np.sin(a) for a in x])
y2 = np.array([np.cos(a) for a in x])
ax1.plot(x1,y1,label=r"$f(x)=\sin(x)$",color='green',linestyle='-',marker='o')
ax2.plot(x2,y2,label=r"$g(x)=\cos(x)$",color='blue',linestyle='--',marker='s')
ax1.set_title("Comportamento da função seno")
ax2.set_title("Comportamento da função cosseno")
```

Múltiplos gráficos em uma única imagem

```
ax1.set_xlabel("Domínio da função")
ax1.set_ylabel("Imagem da função")
ax2.set_xlabel("Domínio da função")
ax2.set_ylabel("Imagem da função")
ax1.legend(loc= 'best')
ax2.legend(loc= 'best')
ax1.grid(True)
ax2.grid(True)
ax1.set_xticks(np.arange(0,10.5,0.5))
ax2.set_xticks(np.arange(0,10.5,0.5))
ax1.set_yticks(np.arange(-2,2.5,0.5))
ax2.set_yticks(np.arange(-2,2.5,0.5))
plt.show()
```

Múltiplos Gráficos em uma única imagem