Matemática: Lições básicas de Python

Doherty Andrade

doherty200@hotmail.com

1 Introdução ao Python: usando Jupyter Notebook

Python é uma linguagem de código aberto, amplamente utilizada e muitas aplicações industriais, acadêmicas e comerciais. Neste minicurso vamos utilizar Python fazer Matemática utilizando o Jupyter Notebook.

O modo mais fácil de utilizar Python é por meio do Jupyter Notebook. Para instalar o Jupyter Notebbok no seu computador use Anaconda. Visiste o site https://www.anaconda.com/distribution/#windows para baixar anaconda para Windows e outras plataformas.

O Anaconda irá instalar em seu computador todas as bibliotecas e recursos necessários para você começar seus projetos em Python e fazer Matemática, Data Science, Machine Learning, Jupyter Notebook, a IDE Spyder, NumPy, Scypy, Sympy, Pandas, Scikit-learn, etc.

O Python foi concebido no final de 1989 por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), nos Países Baixos, como um sucessor da ABC capaz de tratar exceções e prover interface com o sistema operacional Amoeba [8] através de scripts.

O nome é uma homenagem ao grupo de comédia britânico Monty Python, mas essa ideia ficou para trás e hoje todo mundo acredita que o nome originou-se da famosa cobra.

Ao longo do tempo têm sido desenvolvidos pela comunidade de programadores muitas bibliotecas de funções especializadas (módulos) que permitem expandir as capacidades base da linguagem. Entre estes módulos especializados destacam-se os de Matemática:

MatLibplot – para plotagem de gráficos
NumPy– para métodos numéricos
SymPy– para computação simbólica
SciPy – para computação científica
Pandas – para estatística

2 Trabalhando com números

A sintaxe para operações aritméticas em Python é:

+,addition
-,subtraction
*, multiplication
/ , division
** , exponenciação
% , resto da divisão (ou modulo)

// , divisão inteira

Antes de iniciarmos com exemplos vamos chamar os pacotes que eventualmente vamos precisar.

```
In [1]: import numpy as np  # biblioteca básica de matemática
        import sympy as sp
        from sympy import Symbol
        import scipy.linalg   # SciPy biblioteca de algebra linear

        from pylab import *
        import matplotlib
        import matplotlib.pyplot as plt
        import math
```

```
In [2]: 1+1

Out[2]: 2

In [3]: 225-187

Out[3]: 38

In [4]: 3*2

Out[4]: 6

In [5]: 3**2

Out[5]: 9

In [6]: 9**0.5

Out[6]: 3.0

In [7]: 8**(1/3)

Out[7]: 2.0

In [8]: 3/2

Out[8]: 1.5

In [9]: 3//2 # parte inteira da divisão

Out[9]: 1

In [10]: 9%2 # se vc quer apenas o resto da divisão

Out[10]: 1
```

Podemos trabalhar com frações usando

```
In [11]: from fractions import Fraction
         Fraction(3,4)+Fraction(6,4)

Out[11]: Fraction(9, 4)

In [12]: Fraction(3,4)+1.75

Out[12]: 2.5
```

2.1 Observação: (PEMDAS) Python resolve expressões matemáticas

Seguindo o padrão PEMDAS: primeiro calcula parênteses, expoentes, multiplicação, divisão e finalmente, adição e subtração. Vejamos um exemplo

```
In [13]: 3+4*5

Out[13]: 23

In [14]: (3+4)*5

Out[14]: 35
```

Atribuindo nomes a números

```
In [15]: x = 3; y = 4

In [16]: x+y; x*y

Out[16]: 12

In [17]: x+10

Out[17]: 13

In [18]: x**2

Out[18]: 9

In [19]: x**y

Out[19]: 81
```

Diferentes tipos de números: inteiro, aritmética de ponto flutuante (decimal) e complexos.

```
In [20]: type(5)

Out[20]: int

In [21]: type(5.78)

Out[21]: float

In [22]: z = 3 + 4j

In [23]: type(z)

Out[23]: complex

In [24]: w = complex(5,2)

In [25]: type(w)
```

```
Out[25]: complex

In [26]: z + w

Out[26]: (8+6j)

In [27]: z*w

Out[27]: (7+26j)

In [28]: z.real #parte real de z

Out[28]: 3.0

In [29]: z.imag # parte imaginária de z

Out[29]: 4.0

In [30]: z/w

Out[30]: (0.793103448275862+0.48275862068965514j)

In [31]: z.conjugate()

Out[31]: (3-4j)

In [32]: abs(z)

Out[32]: 5.0
```

Comando input: abre uma janela para inserir valor

```
In [33]: a = input()

5


In [34]: a

Out[34]: '5'
```

Comando range:

Quando precisamos que um valor percorra sequencia de valores. Veja exemplo: vamos gerar os valores de 0 a 9.

```
In [35]: list(range(10))

Out[35]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [36]: list(range(2,11))

Out[36]: [2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [37]: list(range(0, 42, 3)) #iniciando em zero e pulando de 3 em 3.

Out[37]: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39]
```

Séries Neste exemplo, vamos calcular a soma parcial de

$$\sum_{i=1}^{100} \frac{1}{i^2}.$$

```
In [38]: n=100
         sum([1/i**2 for i in range(1,n+1)])

Out[38]: 1.6349839001848931
```

Podemos gerar os termos de uma sequência e depois somar. Neste exemplo a sequência é

$$(\frac{1}{2^0}, \frac{1}{2^1}, \frac{1}{2^2}, \ldots, \frac{1}{2^n}, \ldots).$$

```
In [39]: mySequence = [1/2**n  for n in range(0,100)]
         mySum = sum(mySequence)
         mySum

Out[39]: 2.0
```

Definindo funções.
Neste exemplo, definimos a função $f(x) = x^2$.

```
In [40]: def f(x):
             return x ** 2

In [41]: f(10)

Out[41]: 100
```

Função anônima: outro modo de definir uma função.
Neste exemplo, definimos a função $f(x) = x^2$.

```
In [42]: lambda x: x ** 2

Out[42]: <function __main__.<lambda>(x)>

In [43]: (lambda x: x ** 2)(10)

Out[43]: 100
```

Pode-se fazer o mesmo para funções de várias variáveis.

```
In [44]: (lambda x, y, z: (x + y) * z )(10, 20, 2)

Out[44]: 60
```

Função MAP

A função "map" cuja sintaxe é map(f,s) aplica a função f a todos os elementos de uma sequência s.

Aqui observe que Python inicia a contagem com 0 e termina com 9. Isso é padrão.

```
In [45]: def f(x):
             return x ** 2
         lst2 = list(map(f, range(10)))
         lst2

Out[45]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Existe a possibilidade de impressão de números racionais em forma de fração. Veja o exemplo.

```
In [46]: import sympy
         sympy.init_printing()

In [47]: from sympy import Rational

In [48]: a = Rational(2, 5)
         a

Out[48]:
```

$$\frac{2}{5}$$

Trabalhando com símbolos, é preciso do pacote sympy.

```
In [49]: import numpy as np
         import sympy as sp

         from sympy.interactive import printing
         printing.init_printing(use_latex = True)

In [50]: x, y, z = sympy.symbols('x,y,z')
         x + 2*y + 3*z - 2*x + 5*y

Out[50]:
```

$$-x + 7y + 3z$$

Resolvendo EDOs e calculando derivadas.

```
In [51]: from sympy import Symbol, dsolve, Function, Derivative, Eq
         y = Function("y")
         x = Symbol('x')
         y_x = Derivative(y(x), x)
         dsolve(y_x + 5*y(x), y(x))
```

Out[51]:

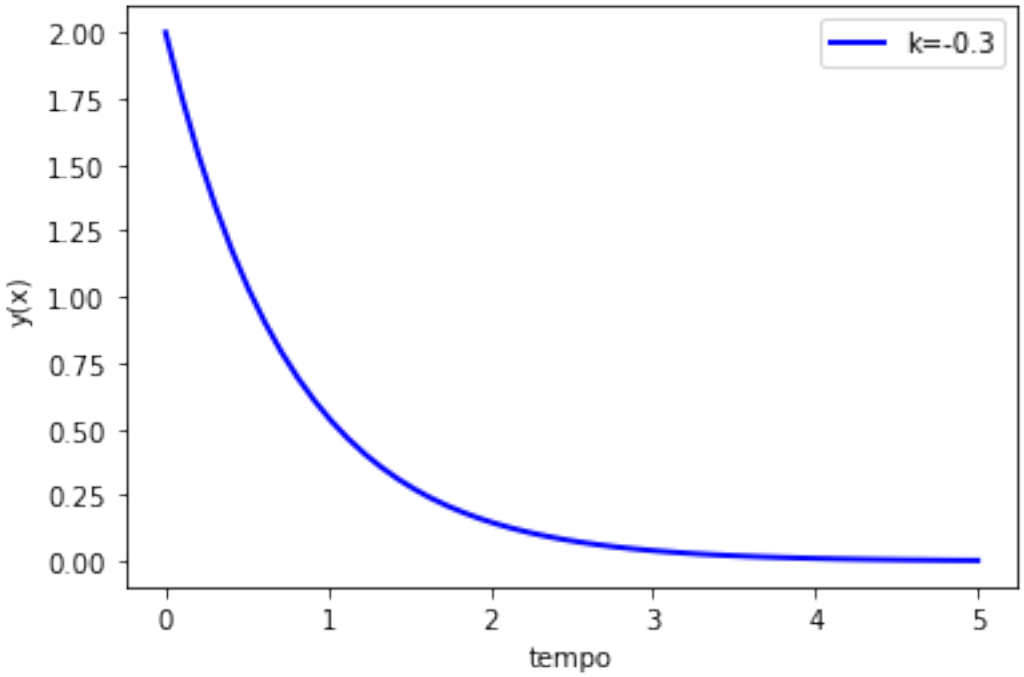
$$y(x) = C_1 e^{-5x}$$

Usando o comando odeint: resolve numericamente PVIs.
Como exemplo, tomemos o PVI dado por Resolver o PVI:

$$\frac{dy}{dx} = -0.3y(x) - y(x)$$

$$y(0) = 2$$

In [52]:
```
import sympy as sp
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# funcao que retorna dy/dx
def modelo(y,x):
    dydx = -0.3 * y -y
    return dydx

# condição inicial
y0 = 2

# intervalo
x = np.linspace(0,5)

# Resolve o PVI

y1 = odeint(modelo,y0,x)


# plot results
plt.plot(x,y1,'b-',linewidth=2,label='k=-0.3')

plt.xlabel('tempo')
plt.ylabel('y(x)')
plt.legend()
plt.show()
```




```
In [53]: from sympy import symbols,solve,Eq
         x, y, z = symbols('x,y,z')
         solve((Eq(3*x+7*y,12*z), Eq(4*x-2*y,5*z)), x, y)
```

Out[53]:

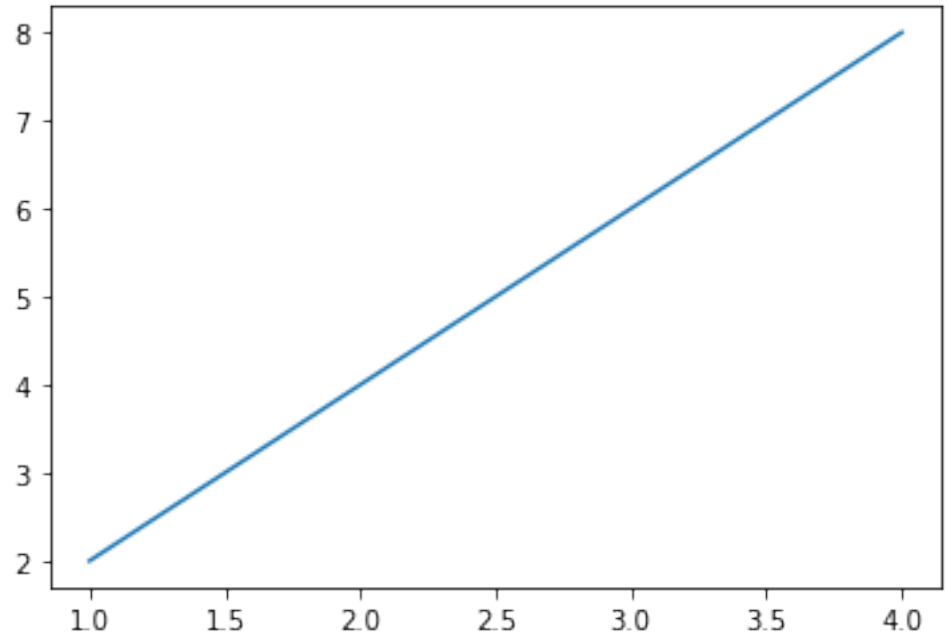
$$\left\{x : \frac{59z}{34}, \quad y : \frac{33z}{34}\right\}$$

3 Criando Gráficos com Matplotlib

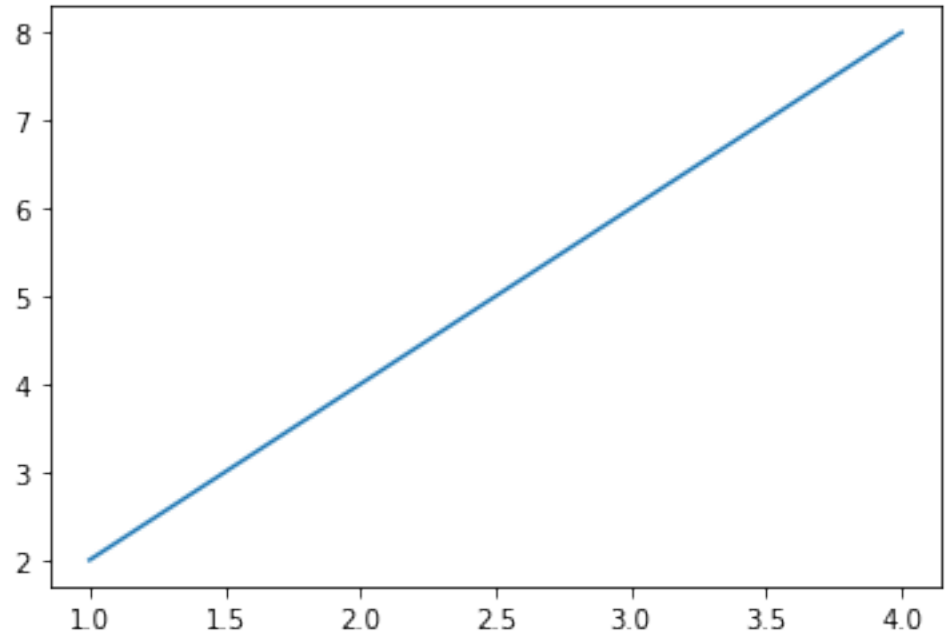
```
In [54]: # criando listas
         x = [1, 2, 3, 4]
         y = [2, 4, 6, 8]
```

```
In [55]: from pylab import plot, show
         plot(x,y)
         show
```

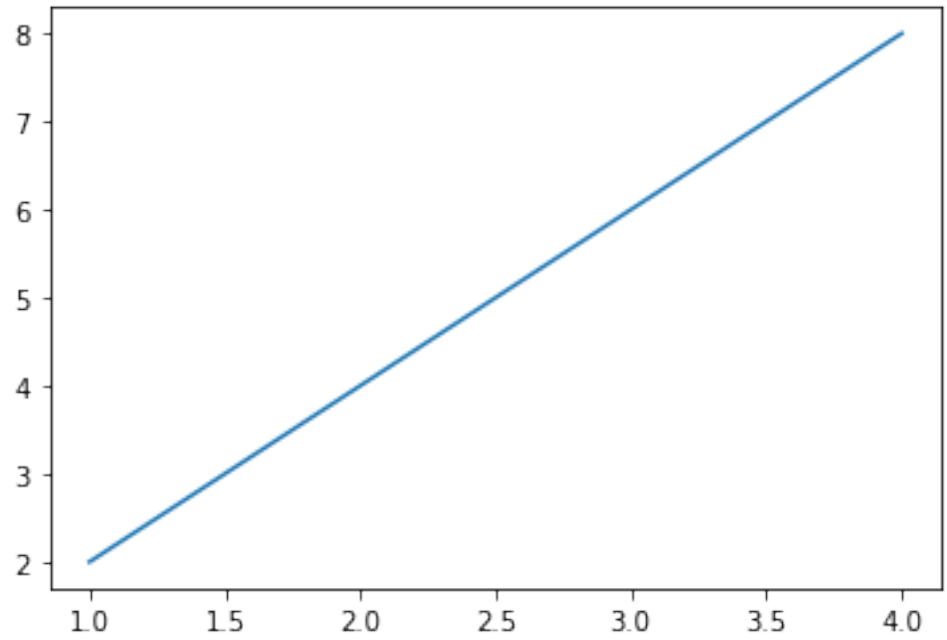
```
Out[55]: <function matplotlib.pyplot.show(*args, **kw)>
```



In [56]:
```
#podemos salvar a figura gerada usando pylab
from pylab import plot, savefig
plot(x,y)
savefig('D:\meugrafico1.png')
```

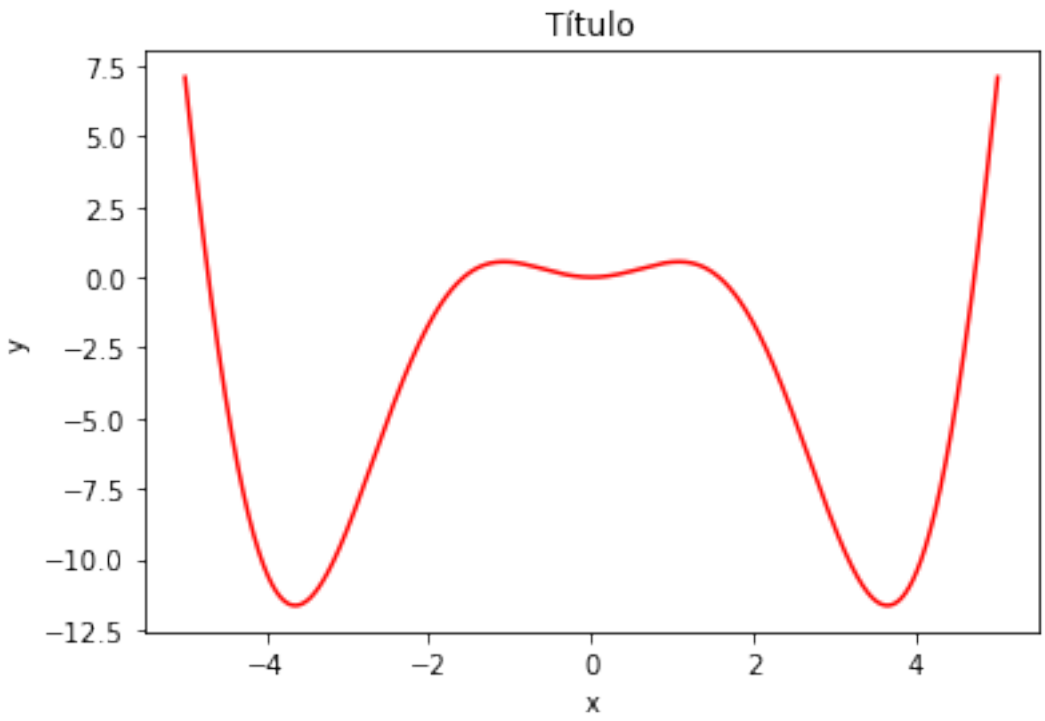


```
In [57]: #podemos salvar a figura usando matplotlib
         from matplotlib import pyplot as plt
         plot(x,y)
         plt.savefig('D:\meugrafico2.png')
         plt.savefig('D:\meugrafico3.pdf')
```

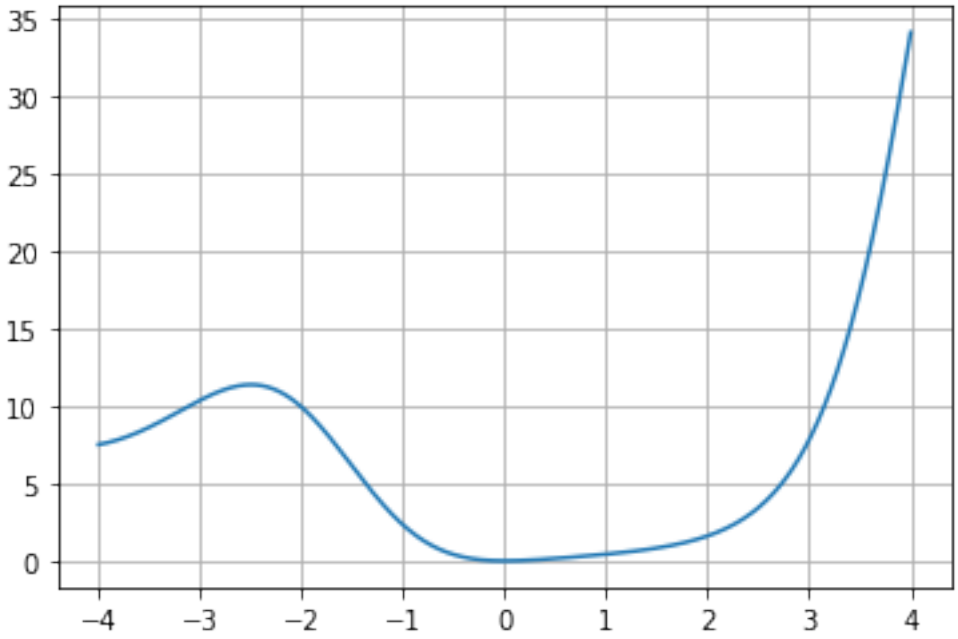


3.0.1 Plotando gráfico de função

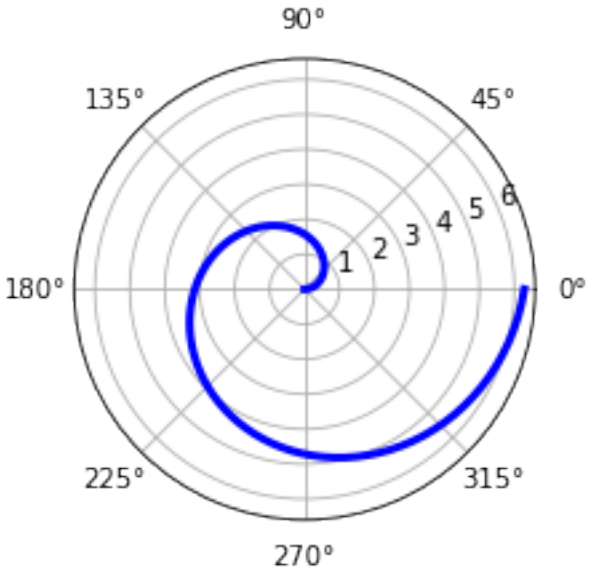
```
In [58]: from pylab import *
         import matplotlib
         import matplotlib.pyplot as plt
         import numpy as np  # biblioteca básica de matemática

In [59]: x = np.linspace(-5, 5,2000)
         y = x ** 2*cos(x)
         figure()
         plot(x, y, 'r')
         xlabel('x')
         ylabel('y')
         title('Título')
         show()
```



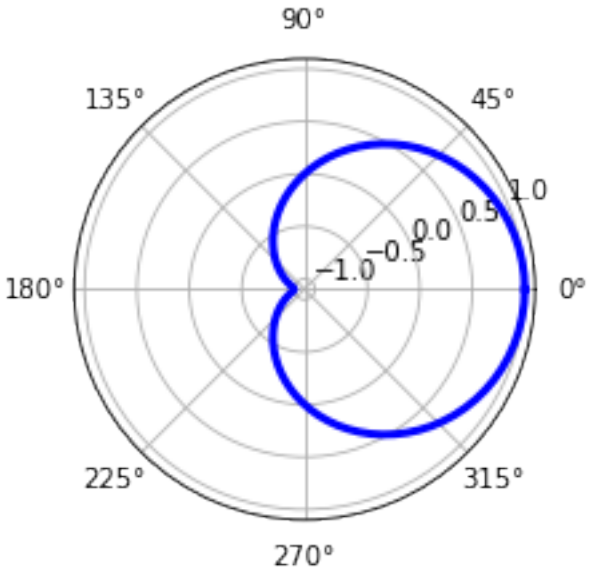

```
In [60]: #podemos incluir um gride
         x = np.linspace(-4,4,1000)
         y = (x**2)*exp(-sin(x))

         plt.plot(x,y)
         grid(True)
         plt.show()
         #tente outras funções
         #y = (x**2)*exp(-x)
```



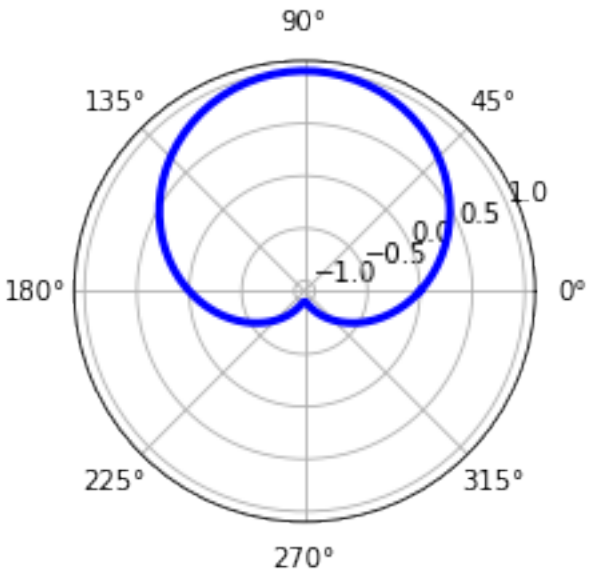
In [61]:
```
# coordenadas polares
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, t, color='blue', lw=3);
## tente (t, cos(t))
## tente (t, sin(t))
## tente (t,exp(-t))
```



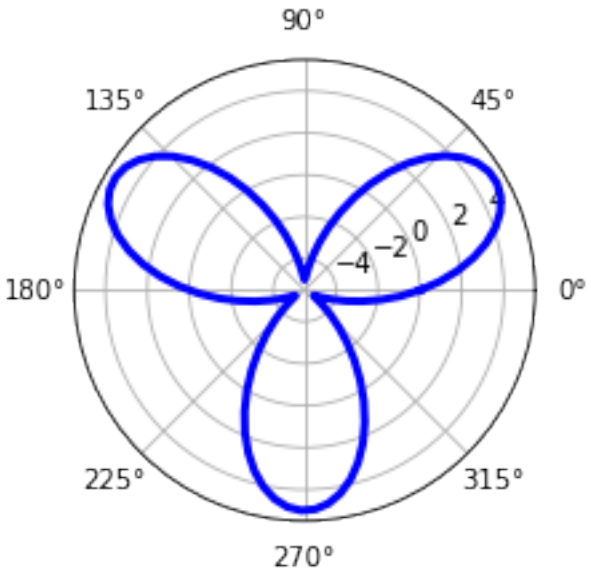
In [62]:
```
# coordenadas polares - cardioide
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, cos(t), color='blue', lw=3);
```



In [63]:
```
# coordenadas polares
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, sin(t), color='blue', lw=3);
```

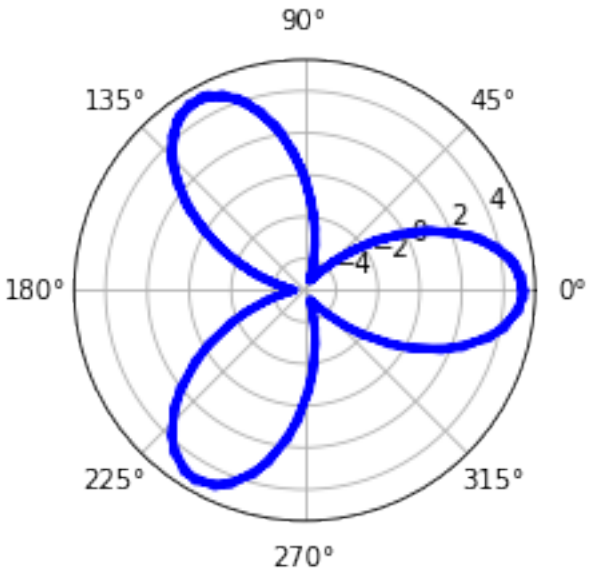


In [64]:
```
# coordenadas polares- rosa de 3 folhas
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, 5*sin(3*t), color='blue', lw=3);
```

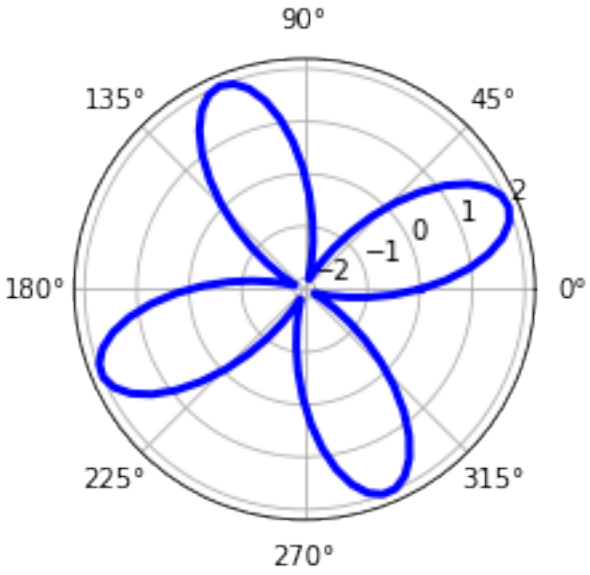


In [65]:
```
# coordenadas polares- rosa de 3 folhas
fig = plt.figure()
```

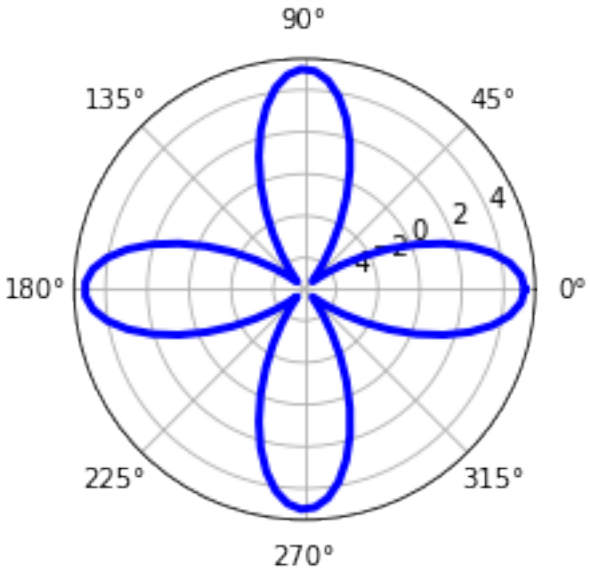
```
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 5 * np.pi, 100)
ax.plot(t, 5*cos(3*t), color='blue', lw=3);
```



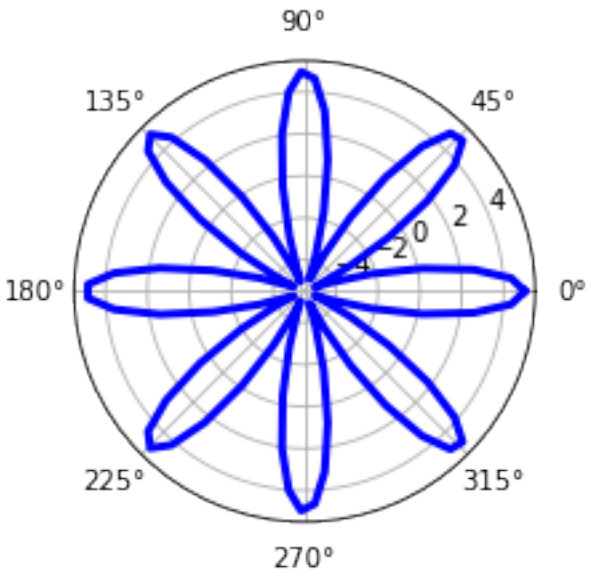
In [66]:
```
# coordenadas polares- rosa de 4 folhas
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, 2*sin(4*t), color='blue', lw=3);
```



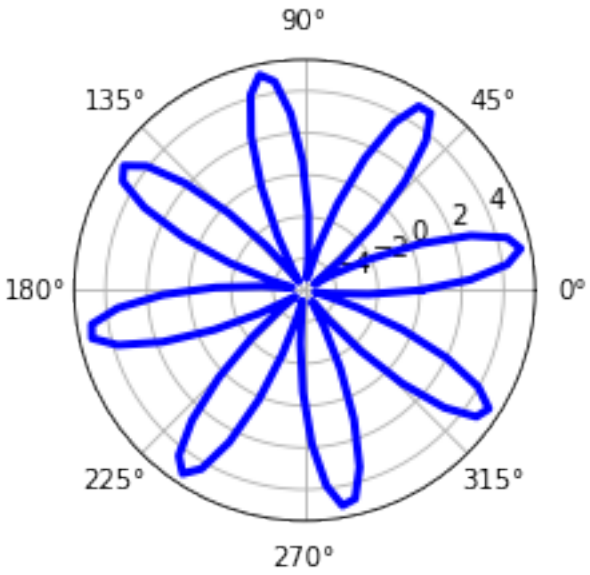
In [67]:
```
# coordenadas polares- rosa de 4 folhas
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, 5*cos(4*t), color='blue', lw=3);
```



In [68]:
```
# coordenadas polares- rosa de 8 folhas
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, 5*cos(8*t), color='blue', lw=3);
```



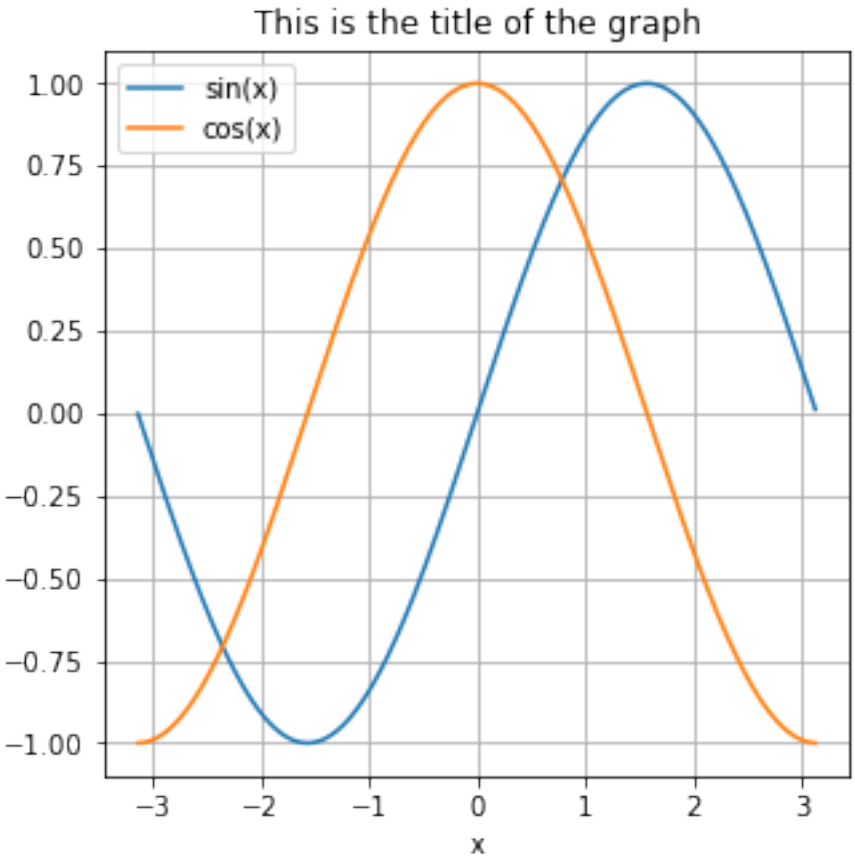
In [69]:
```
# coordenadas polares- rosa de 8 folhas
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, 5*sin(8*t), color='blue', lw=3);
```



In [70]:
```
import pylab
import numpy as N
```

```
x = N.arange(-3.14, 3.14, 0.01)
y1 = N.sin(x)
y2 = N.cos(x)
pylab.figure(figsize =(5 , 5))
pylab.plot(x, y1, label='sin(x)')
pylab.plot(x, y2, label='cos(x)')
pylab.legend()
pylab.grid()
pylab.xlabel('x')
pylab.title('This is the title of the graph')
```

```
Out[70]: Text(0.5, 1.0, 'This is the title of the graph')
```



3.1 Criando programinhas que fazem Matemática

Python utiliza identação. Portanto cuidado.

```
In [71]: #observe onde termina
         for i in range(1,5):
             print(i)
```

```
1
2
3
4
```

```
In [72]: #observe onde começa
         for i in range(5):
             print(i)
```

```
0
1
2
3
4
```

```
In [73]: #pulando de 2 em 2. Escolha outros saltos (steps)
         for i in range(1,10,2):
             print(i)
```

```
1
3
5
7
9
```

```
In [74]: mySequence = [1/2**n  for n in range(0,100)]
         mySum = sum(mySequence)
         mySum
```

```
Out[74]:
```

2.0

3.1.1 Divisores de um número

```
In [75]: #se o resto da divisao inteira for zero, achamos um divisor para o numero
         def fatores(n):
             for i in range(1,n+1):
                 if n % i == 0:
                     print(i)
```

```
In [76]: fatores(123)
```

```
1
3
41
123
```

3.1.2 Fatorial de um número: while

```
In [77]: #estamos usando o comando input e o comando while
         def Fatorial():
             "Calcula N! = N(N-1) ... (2)(1) for N >= 1."
             n = int(input("Entre com o número inteiro n: "))
             fat = 1
             i = 1
             while i <= n:
                 fat = fat*i
                 i = i + 1

             print("O valor de %d! é =" %n, fat)
```

```
In [78]: Fatorial()
```

```
Entre com o número inteiro n: 6
O valor de 6! é = 720
```

```
In [79]: #usando for
         def fatorial(N):
             "Calcula N! = N(N-1) ... (2)(1) for N >= 1."
             # Inicialize com 1
             produto = 1
             for n in range(2,N + 1):
                 # atualiza
                 produto = produto * n
             return produto
```

```
In [80]: fatorial(5)
```

```
Out[80]:
```

120

Podemos usar o fatorial para aproximar o valor de e usando a série de Taylor de e^x:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Como exemplo, vamos calular a 100ésima soma parcial da série com $x = 1$:

```
In [81]: sum([1/fatorial(k) for k in range(0,101)])
```

```
Out[81]:
```

2.718281828459045

```
In [82]: for n in range(0,5,1):
             print(2**n -1)
```

```
0
1
3
7
15
```

4 Somando: usando while

```
In [83]: def Soma(n):
             """ Retorna a soma de 1+2+3+ ...+ n """

             soma  = 0
             numero = 1
             while numero <= n:
                 soma = soma + numero
                 numero = numero + 1
             return soma
```

```
In [84]: Soma(4)
```

Out[84]:

10

```
In [85]: Soma(100)
```

Out[85]:

5050

Pequena mudança na soma

```
In [86]: def soma2(n):
             """ Retorna a soma de 1+2^2+3^2+ ...+ n^2 """
             soma2  = 0
             numero = 1
             while numero <= n:
                 soma2 = soma2 + numero**2
                 numero = numero + 1
             return soma2
```

```
In [87]: soma2(5)
```

Out[87]:

55

5 Sequência de Fibonacci

Asequência de Fibonacci é a sequência 0, 1,1, 2, 3, 5, 8 , 13,..., onde o um termo é soma dos dois imediatamente anteriores. Isto é,

$$a_n = a_{n-1} + a_{n-2}.$$

```
In [88]: #usando for
         fibonacci = [0,1]
         for n in range(2,25):
             fibonacci_n = fibonacci[n-1] + fibonacci[n-2]
             fibonacci.append(fibonacci_n)
             print(fibonacci)
```

```
[0, 1, 1]
[0, 1, 1, 2]
[0, 1, 1, 2, 3]
[0, 1, 1, 2, 3, 5]
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 13]
[0, 1, 1, 2, 3, 5, 8, 13, 21]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946
```

```
In [89]: #melhorando com o input e o for
         fibonacci = [0,1]
         n = int(input("Digite um número natural maior do que 2: "))
         for k in range(2,n):
             fibonacci_k = fibonacci[k-1] + fibonacci[k-2]
             fibonacci.append(fibonacci_k)
             print(fibonacci)
```

```
Digite um número natural maior do que 2: 9
[0, 1, 1]
[0, 1, 1, 2]
```

```
[0, 1, 1, 2, 3]
[0, 1, 1, 2, 3, 5]
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 13]
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

```
In [90]: #usando if
         # par ou impar
         def parouimpar():
             n = int(input("Digite um número natural: "))

             if n % 2 == 0:  # se n é múltiplo de 2
                 print(n, "é par")
             if n % 2 != 0:  # se n não é múltiplo de 2
                 print(n , "é impar")
```

```
In [91]: parouimpar()

Digite um número natural: 343
343 é impar
```

5.0.1 Melhorando o código

```
In [92]: #múltiplo de 7
         def mult7():
             n = int(input("Digite um número: "))

             if n % 7 == 0:  # se n é múltiplo de 7
                 print(n, "é múltiplo de 7")
             if n % 7 != 0:  # se n não é múltiplo de 7
                 print(n , " não é múltiplo de 7")
             print("fim.")

         #-----
         if __name__ == '__main__':
             mult7()

Digite um número: 343
343 é múltiplo de 7
fim.
```

6 Fórmula de Ramanujan

A fórmula de Ramanujan para π é uma série numérica maravilhosa para expressar π que converge muito rapidamente:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{99^2} \sum_{k=0}^{\infty} \frac{(4k)!}{k!^4} \frac{1103 + 26390k}{396^{4k}}$$

Para encontrar uma aproximação para π usamos um número finito de termos, a seguir usando os três primeiros termos da série para obter uma boa aproximação para π:

$$\pi \approx \frac{99^2}{2\sqrt{2}} \frac{1}{\left(1103 + 4! \frac{1103+26390}{396^4} + \frac{8!}{2^4} \frac{1103+26390(2)}{396^8}\right)}$$

In [93]:
```
#com 3 termos na soma
99**2 / (2 * 2**0.5) / (1103 + 4*3*2 * (26390 + 1103) / 396**4+ 8*7*6*5*4*3*2 / 2**4 *
```

Out[93]:

3.141592653589793

In [94]:
```
#Melhorando um pouco mais-- calculando com qualquer soma parcial tomando n qualquer
import numpy as np
import math
n = 10
(99**2)/(2*math.sqrt(2))*(1/(sum([math.factorial(4*k)*(1103+26390*k)/
                                  (math.factorial(k)**4*396**(4*k)) for k in range(0,n+1)])))
```

Out[94]:

3.141592653589793

In [95]:
```
#definindo uma funçao que retorna a n-ésima soma de Ramanujan
def ramanujan(n):
    return (99**2)/(2*math.sqrt(2))*(1/(sum([math.factorial(4*k)*(1103+26390*k)/(math.f
```

In [96]:
```
ramanujan(20)
```

Out[96]:

3.141592653589793

7 If - else

If-else: Um if-else é utilizado quando temos apenas duas alternativas. Quando o número de alternativas é maior, podemos aninhar comandos if-else. Por exemplo, considere o problema de ler a nota de um aluno para verificar se ele está reprovado, está de recuperação ou foi aprovado. Suponha que as notas são números inteiros entre 0 e 100. Um aluno está reprovado se sua nota e menor que 30, está de recuperação se sua nota é um inteiro entre 30 e 49 e está aprovado se sua nota é pelo menos 50. Uma solução aninhando comandos if-else seria:

```
In [97]: def main():
             nota = int(input("Digite uma nota inteira entre 0 e 100: "))

             if nota < 30:
                 print("Reprovado")
             else:
                 if nota < 60:
                     print("Recuperacao")
                 else:
                     print("Aprovado")

             print("fim.")

         #-----
         if __name__ == '__main__':
             main()
```

```
Digite uma nota inteira entre 0 e 100: 45
Recuperacao
fim.
```

elif:

elif é apenas uma contração do else if que torna mais claro o tratamento das várias alternativas, encadeando as condições. Blocos de elif podem ser repetido várias vezes.

```
In [98]: def main():
             nota = int(input("Digite uma nota entre 0 e 100: "))

             if nota < 30:
                 print("Reprovado")
             elif nota < 50:
                 print("Recuperacao")
             else:
                 print("Aprovado")

             print("fim.")

         #-----
         if __name__ == '__main__':
             main()
```

```
Digite uma nota entre 0 e 100: 45
Recuperacao
fim.
```

7.0.1 Testando numeros primos

```
In [99]: import math
         def ehprimo(n):
             for d in range(2,int(math.sqrt(n)+1)):
                 if n%d == 0:
                     return False
             return True

In [100]: ehprimo(6173)

Out[100]: True
```

8 Primos de Mersenne

São números primos da forma $M_p = 2^p - 1$, onde p é primo. Mesmo que p seja primo, M_p pode não ser primo, é o caso de $p = 11$.

O conhecimento de números primos grandes é a base da segurança da criptografia.

```
In [101]: #precisa do programa ehprimo acima
          # gerando numeros de Mersenne
          def mersene(n):
              if ehprimo(n):
                  print ( 2**n -1 )
              else:
                  return False

In [102]: mersene(11)

2047


In [103]: #precisa do programa ehprimo acima
          # testando se um número de mersenne é primo
          def merseneprimo(n):
              if ehprimo(2**n-1):
                  print(2**n-1,',o numero de Mersene é primo')
              else:
                  print(2**n-1,',o numero de Mersene NÃO é primo')

In [104]: merseneprimo(11)

2047 ,o numero de Mersene NÃO é primo


In [105]: #precisa do programa ehprimo acima
          # testando se um número n e o numero de mersenne corresponde sào primos
          def merseneprimo2(n):
              if ehprimo(n):
```

```
                if ehprimo(2**n-1):
                    print(n, 'e,', 2**n-1,',o numero e o Mersene sao primos')
                else:
                    print(2**n-1,',o numero de Mersene NÃO é primo')
            else:
                print(n,',  NÃO é primo')
```

```
In [106]: merseneprimo2(5)
```

```
5 e, 31 ,o numero e o Mersene sao primos
```

```
In [107]: #precisa do programa ehprimo acima
          # testando se os números até n e o numero de mersenne corresponde sào primos
          def merseneprimo3(m):
              for n in range(2,m+1):
                  if ehprimo(n):
                      if ehprimo(2**n-1):
                          print(n, 'e,', 2**n-1,',o numero e o Mersene sao primos')
                      else:
                          print(n,',', 2**n-1,',o numero de Mersene NÃO é primo')
                  else:
                      print(n,', o número  NÃO é primo')
```

```
In [108]: merseneprimo3(20)
```

```
2 e, 3 ,o numero e o Mersene sao primos
3 e, 7 ,o numero e o Mersene sao primos
4 , o número  NÃO é primo
5 e, 31 ,o numero e o Mersene sao primos
6 , o número  NÃO é primo
7 e, 127 ,o numero e o Mersene sao primos
8 , o número  NÃO é primo
9 , o número  NÃO é primo
10 , o número  NÃO é primo
11 , 2047 ,o numero de Mersene NÃO é primo
12 , o número  NÃO é primo
13 e, 8191 ,o numero e o Mersene sao primos
14 , o número  NÃO é primo
15 , o número  NÃO é primo
16 , o número  NÃO é primo
17 e, 131071 ,o numero e o Mersene sao primos
18 , o número  NÃO é primo
19 e, 524287 ,o numero e o Mersene sao primos
20 , o número  NÃO é primo
```

Criando uma função que recebe Fahrenheit e devolve Celsius

```
In [109]: #graus Fahrenheit em Celsius
          def Celsius(F):
              C = (F-32)*(5/9)
              print(C, 'graus Celsius')

In [110]: Celsius(98.6)

37.0 graus Celsius
```

Criando uma função que recebe polegadas e devolve metros.

```
In [111]: #Polegadas em metros-- cada polegada mede 2.54 cm
          def metro( ):
              p = float(input("Digite a medida em polegadas:"))

              m = p*0.0254
              print(m, ' metros')
          #-----
          if __name__ == '__main__':
              metro( )

Digite a medida em polegadas:456
11.5824  metros
```

9 Conjectura de Collatz

Seja a um inteiro positivo e a sequência definida recursivamente do seguinte modo: se a é par tome a sua metade, e se a é impar tome $3a + 1$. Repita o processo até atingir o número 1. A conjecura de Collatz afirma que esta sequencia sempre atingirá 1.

```
In [112]: def collatz(a):
              "Determine os elementos da sequencia de Collatz inciando em a e termiando em 1."
              # Initializa a  lista com  o primeiro valor a
              sequence = [a]
              # calcula os valores até 1
              while sequence[-1] > 1:
                  # verifica se o ultimo elemento da lista é par
                  if sequence[-1] % 2 == 0:
                      # calculo e acrescenta o novo valor
                      sequence.append(sequence[-1] // 2)
                  else:
                      # calculo e acrescenta o novo valor
                      sequence.append(3*sequence[-1] + 1)
              return sequence
```

```
In [113]: collatz(67)
```

```
Out[113]:
```

```
[67, 202, 101, 304, 152, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40,
```

Razão Áurea

```
In [114]: #razão áurea

          def aurea(x,y):
              a = x/y
              b = (x+y)/x
              if a == b:
                  print('há razao aurea')
              else:
                  print('não ha razao aurea')
```

```
In [115]: aurea(1,2)
```

```
não ha razao aurea
```

```
In [116]: f = 1.618033988749895
          aurea(3*f,3)
```

```
há razao aurea
```

9.1 Fórmula de Baskhara

```
In [117]: import numpy as np

          def bask(a,b,c):
              if (b**2-4*a*c)  >= 0:
                  x1 = (-b+np.sqrt(b**2-4*a*c))/(2*a)
                  x2 = (-b-np.sqrt(b**2-4*a*c))/(2*a)
                  print('as raizes reais sao:,',x1,x2)
              else:
                  print('as raizes sao complexas')
```

Determinar utilizando Báskhara as soluções de

$$x^2 - x - 1 = 0.$$

```
In [118]: bask(1,-1,-1)
```

```
as raizes reais sao:, 1.618033988749895 -0.6180339887498949
```

```
In [119]: #geral
          from cmath import sqrt
          import numpy as np
          def bask2(a,b,c):
              if (b**2-4*a*c)  >= 0:
                  x1 = (-b+np.sqrt(b**2-4*a*c))/(2*a)
                  x2 = (-b-np.sqrt(b**2-4*a*c))/(2*a)
                  print('as raizes reais sao:,',x1,'e', x2)
              else:
                  x1 = (-b+ sqrt(b**2-4*a*c))/(2*a)
                  x2 = (-b- sqrt(b**2-4*a*c))/(2*a)
                  print('as raizes sao complexas:,', x1,'e', x2)

In [120]: bask2(1,3,9)

as raizes sao complexas:, (-1.5+2.598076211353316j) e (-1.5-2.598076211353316j)
```

10 Um pouco de estatística

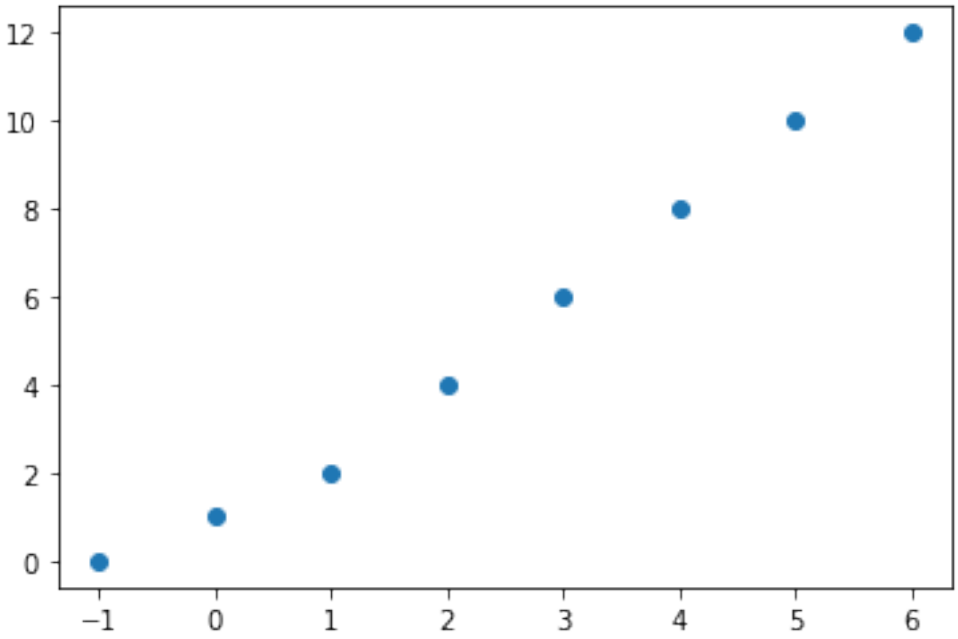
```
In [121]: lista = [1,2,3,4,5,6]

In [122]: sum(lista) # somando os elementos da lista

Out[122]: 21

In [123]: len(lista) # comprimento da lista

Out[123]:
```

6

```
In [124]: media = sum(lista)/len(lista)
          media

Out[124]:
```

3.5

```
In [125]: x = [1, 2, 3, 4, 5 ,6, 0,-1]
          y = [2, 4, 6, 8, 10, 12, 1,0]
          import matplotlib.pyplot as plt
          plt.scatter(x, y)
          plt.show()
```



11 Álgebra e Matemática Simbólica com Sympy

```
In [126]: from sympy import Symbol
          x = Symbol('x')

In [127]: x + x + 1
```

Out[127]:

$$2x + 1$$

```
In [128]: x= Symbol('x')
          y = Symbol('y')
          z = Symbol('z')

In [129]: s = x*y+x*y
          s
```

Out[129]:

$$2xy$$

```
In [130]: p = (x+2)*(y+3)# Sympy multiplica e depois simplifica--ahahaha
          p
```

Out[130]:

$$(x+2)(y+3)$$

In [131]:
```
from sympy import factor
expressao = x**2- y**2
factor(expressao)
```

Out[131]:

$$(x-y)(x+y)$$

Exemplo: fatorar a expressão

$$x^3 + 3x^2y + 3xy^2 + y^3.$$

In [132]:
```
expr1 = x**3 + 3*x**2*y + 3*x*y**2 + y**3
expr2 = factor(expr1)
```

In [133]:
```
from sympy import *
```

In [134]:
```
simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)) ## comando simplify
```

Out[134]:

$$x - 1$$

In [135]:
```
expand((x + 1)**2)
```

Out[135]:

$$x^2 + 2x + 1$$

In [136]:
```
from sympy import pprint ## ficou melhor?Pretty Printing
pprint(expr1)
```

```
 3      2          2    3
x  + 3·x ·y + 3·x·y  + y
```

In [137]:
```
expr3 = x*x + x*y + x*y + y*y ## substituindo valores nas expressores
res = expr3.subs({x:1, y:2})
```

In [138]:
```
res
```

Out[138]:

9

In [139]:
```
expr3.subs({x:1-y})
```

Out[139]:

$$y^2 + 2y(-y+1) + (-y+1)^2$$

11.0.1 Resolvendo equações

```
In [140]: from sympy import Symbol, solve
          x = Symbol('x')
          expr = x - 5 - 7
          solve(expr)
```

Out[140]:

$$[12]$$

```
In [141]: from sympy import solve
          x = Symbol('x')
          expr = x**2 + 5*x + 4
          solve(expr, dict=True)
```

Out[141]:

$$[\{x : -4\}, \quad \{x : -1\}]$$

```
In [142]: x=Symbol('x')
          expr = x**2 + x + 1
          solve(expr, dict=True)
```

Out[142]:

$$\left[\left\{x : -\frac{1}{2} - \frac{\sqrt{3}i}{2}\right\}, \quad \left\{x : -\frac{1}{2} + \frac{\sqrt{3}i}{2}\right\}\right]$$

```
In [143]: x = Symbol('x')
          y = Symbol('y')
          expr1 = 2*x + 3*y - 6
          expr2 = 3*x + 2*y - 12
```

```
In [144]: solve((expr1, expr2), dict=True)
```

Out[144]:

$$\left[\left\{x : \frac{24}{5}, \quad y : -\frac{6}{5}\right\}\right]$$

11.0.2 Derivada

```
In [145]: from sympy import *
          x, y, z = symbols('x y z')
```

```
In [146]: diff(cos(x), x)
```

Out[146]:

$$-\sin(x)$$

```
In [147]: diff(x**4, x, x, x)
```

Out[147]:

$$24x$$

```
In [148]: expr = exp(x*y*z)
          diff(expr, x, y, y, z, z, z)
```

Out[148]:

$$x^2y\left(x^3y^3z^3 + 11x^2y^2z^2 + 30xyz + 18\right)e^{xyz}$$

11.0.3 Integral

```
In [149]: integrate(cos(x), x)
```

Out[149]:

$$\sin\left(x\right)$$

```
In [150]: integrate(cos(x), (x, 0, pi/2))
```

Out[150]:

$$1$$

```
In [151]: integrate(exp(-x), (x, 0, oo))
```

Out[151]:

$$1$$

11.0.4 Integral dupla

```
In [152]: integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo))
```

Out[152]:

$$\pi$$

11.0.5 Limites

```
In [153]: limit(sin(x)/x, x, 0)
```

Out[153]:

$$1$$

```
In [154]: limit(x*cos(x), x, 0, '+')
```

Out[154]:

$$0$$

In [155]:
```
limit(sin(x)/x, x, 0, '-')
```

Out[155]:

$$1$$

In [156]:
```
solveset(Eq(x**2 - 1, 8), x)## resolvendo equações
```

Out[156]:

$$\{-3, 3\}$$

In [157]:
```
nonlinsolve([x*y - 1, x - 2], x, y)
```

Out[157]:

$$\left\{\left(2, \quad \frac{1}{2}\right)\right\}$$

In [158]:
```
from sympy import symbols,solve,Eq
x, y, z = symbols('x,y,z')
solve((Eq(3*x+7*y,12*z), Eq(4*x-2*y,5*z)), x, y)
```

Out[158]:

$$\left\{x : \frac{59z}{34}, \quad y : \frac{33z}{34}\right\}$$

In [159]:
```
import sympy                                    # chama o sympy
x, y, z = sympy.symbols('x, y, z')              # cria os simbolos
eq = x - x ** 3                                 # define a equaçao
sympy.solve(eq, x)                              # resolsolve a eq = 0
```

Out[159]:

$$[-1, \quad 0, \quad 1]$$

11.0.6 Matrizes

In [160]:
```
from sympy import *
init_printing(use_unicode=True)
```

In [161]:
```
Matrix([[1, -1], [3, 4], [0, 2]])
```

Out[161]:

$$\begin{bmatrix} 1 & -1 \\ 3 & 4 \\ 0 & 2 \end{bmatrix}$$

In [162]:
```
Matrix([1, 2, 3])
```

Out[162]:

$$\begin{bmatrix}1\\2\\3\end{bmatrix}$$

In [163]:
```
M = Matrix([[1, 2, 3], [-2, 0, 4]])
M
```

Out[163]:

$$\begin{bmatrix}1 & 2 & 3\\-2 & 0 & 4\end{bmatrix}$$

In [164]:
```
M.shape
```

Out[164]:

$$(2,\ 3)$$

In [165]:
```
M = Matrix([[1, 2, 3], [-2, 0, 4]])
N = Matrix([[0, 3,5], [0, 7,-3]])
M + N
```

Out[165]:

$$\begin{bmatrix}1 & 5 & 8\\-2 & 7 & 1\end{bmatrix}$$

In [166]:
```
M = Matrix([[1, 3], [4, 5]])
N = Matrix([[3, 1], [0, 7]])
```

In [167]:
```
M * N
```

Out[167]:

$$\begin{bmatrix}3 & 22\\12 & 39\end{bmatrix}$$

In [168]:
```
2*M
```

Out[168]:

$$\begin{bmatrix}2 & 6\\8 & 10\end{bmatrix}$$

In [169]:
```
M**2,

## tente tb M*M
```

Out[169]:

$$\left(\begin{bmatrix} 13 & 18 \\ 24 & 37 \end{bmatrix}\right)$$

In [170]:
```
M.T # transposta
```

Out[170]:

$$\begin{bmatrix} 1 & 4 \\ 3 & 5 \end{bmatrix}$$

In [171]:
```
M.det()
```

Out[171]:

$$-7$$

11.0.7 Matriz escalonada

para por uma matriz na forma escalonada use o comando rref.

O retorna é a matriz na forma escada, juntamente com os indices das colunas onde ocorre pivot.

In [172]:
```
M = Matrix([[1, 0, 1, 3], [2, 3, 4, 7], [-1, -3, -3, -4]])
M
```

Out[172]:

$$\begin{bmatrix} 1 & 0 & 1 & 3 \\ 2 & 3 & 4 & 7 \\ -1 & -3 & -3 & -4 \end{bmatrix}$$

In [173]:
```
M.rref()
```

Out[173]:

$$\left(\begin{bmatrix} 1 & 0 & 1 & 3 \\ 0 & 1 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad (0, \quad 1)\right)$$

Autovalores e autovetores:

In [174]:
```
M = Matrix([[3, -2,  4, -2], [5,  3, -3, -2], [5, -2,  2, -2], [5, -2, -3,  3]])
M.eigenvals() # retorna autovalores com multiplicidade algébrica
```

Out[174]:

$$\{-2 : 1, \quad 3 : 1, \quad 5 : 2\}$$

In [175]:
```
M.eigenvects()#  retorna seus autovetores
```

Out[175]:

$$\left[\left(-2,\quad 1,\quad \left[\begin{bmatrix}0\\1\\1\\1\end{bmatrix}\right]\right),\quad \left(3,\quad 1,\quad \left[\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\right]\right),\quad \left(5,\quad 2,\quad \left[\begin{bmatrix}1\\1\\1\\0\end{bmatrix},\ \begin{bmatrix}0\\-1\\0\\1\end{bmatrix}\right]\right)\right]$$

11.0.8 Diagonalização

Para diagonalizar uma matriza, use o comando diagonalize. Sympy retorna um par (P,D), onde D diagonal e $M = PDP^{-1}$.

```
In [176]: P, D = M.diagonalize()
```

```
In [177]: P
```

Out[177]:

$$\begin{bmatrix}0 & 1 & 1 & 0\\1 & 1 & 1 & -1\\1 & 1 & 1 & 0\\1 & 1 & 0 & 1\end{bmatrix}$$

```
In [178]: D
```

Out[178]:

$$\begin{bmatrix}-2 & 0 & 0 & 0\\0 & 3 & 0 & 0\\0 & 0 & 5 & 0\\0 & 0 & 0 & 5\end{bmatrix}$$

```
In [179]: R =P*D*P**-1  ## tem que ser igual a M
          print(R)
```

```
Matrix([[3, -2, 4, -2], [5, 3, -3, -2], [5, -2, 2, -2], [5, -2, -3, 3]])
```

11.0.9 Polinômio característico

```
In [180]: lamda = symbols('lamda')
          p = M.charpoly(lamda)
```

```
In [181]: p
```

Out[181]:

$$\text{PurePoly}\left(\lambda^4 - 11\lambda^3 + 29\lambda^2 + 35\lambda - 150, \lambda, domain = \mathbb{Z}\right)$$

```
In [182]: factor(p)
```

Out[182]:

$$(\lambda - 5)^2 (\lambda - 3) (\lambda + 2)$$

In [183]: M.inv()# *inversa de matriz*

Out[183]:

$$\begin{bmatrix} \frac{1}{3} & \frac{2}{15} & -\frac{4}{15} & \frac{2}{15} \\ \frac{5}{6} & \frac{1}{3} & -\frac{29}{30} & \frac{2}{15} \\ \frac{5}{6} & \frac{2}{15} & -\frac{23}{30} & \frac{2}{15} \\ \frac{5}{6} & \frac{2}{15} & -\frac{29}{30} & \frac{1}{3} \end{bmatrix}$$

12 Decomposição LU

```
In [184]: #Decomposição LU
          import scipy
          import scipy.linalg   # SciPy Linear Algebra Library

          A = scipy.array([ [7, 3, -1, 2], [3, 10, 1, -4], [-1, 1, 8, -1], [2, -4, -1, 9] ])
          P, L, U = scipy.linalg.lu(A)


          print('A=',A)

          print('P=',P)

          print('L=',L)

          print('U=',U)
```

```
A= [[ 7  3 -1  2]
 [ 3 10  1 -4]
 [-1  1  8 -1]
 [ 2 -4 -1  9]]
P= [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
L= [[ 1.          0.          0.          0.        ]
 [ 0.42857143  1.          0.          0.        ]
 [-0.14285714  0.16393443  1.          0.        ]
 [ 0.28571429 -0.55737705  0.01075269  1.        ]]
U= [[ 7.          3.         -1.          2.        ]
 [ 0.          8.71428571  1.42857143 -4.85714286]
 [ 0.          0.          7.62295082  0.08196721]
 [ 0.          0.          0.          5.72043011]]
```

13 Decomposição Cholesky

```
In [185]: #Decomposição Cholesky
          import pprint
          import scipy
          import scipy.linalg   # SciPy Linear Algebra Library

          A = scipy.array([[20, 3, 4, 8], [3, 20, 5, 1], [4, 5, 30, 7], [8, 1, 7, 25]])
          L = scipy.linalg.cholesky(A, lower=True)
          U = scipy.linalg.cholesky(A, lower=False)

          print('A=')
          pprint.pprint(A)

          print('L=')
          pprint.pprint(L)

          print('Lt=U=')
          pprint.pprint(U)
```

```
A=
array([[20,  3,  4,  8],
       [ 3, 20,  5,  1],
       [ 4,  5, 30,  7],
       [ 8,  1,  7, 25]])
L=
array([[ 4.47213595,  0.        ,  0.        ,  0.        ],
       [ 0.67082039,  4.42153819,  0.        ,  0.        ],
       [ 0.89442719,  0.9951288 ,  5.31128221,  0.        ],
       [ 1.78885438, -0.04523313,  1.02517859,  4.5548834 ]])
Lt=U=
array([[ 4.47213595,  0.67082039,  0.89442719,  1.78885438],
       [ 0.        ,  4.42153819,  0.9951288 , -0.04523313],
       [ 0.        ,  0.        ,  5.31128221,  1.02517859],
       [ 0.        ,  0.        ,  0.        ,  4.5548834 ]])
```

14 Otimização

Otimização

Em geral precisamos encontrar o maximo e o minimo de uma função particular f escalar. As rotinas de otimização são tipicamente de minimização. Para maximizar invertemos o sinal de f definindo uma nova função $g(x)=-f(x)$ e minimizamos g.

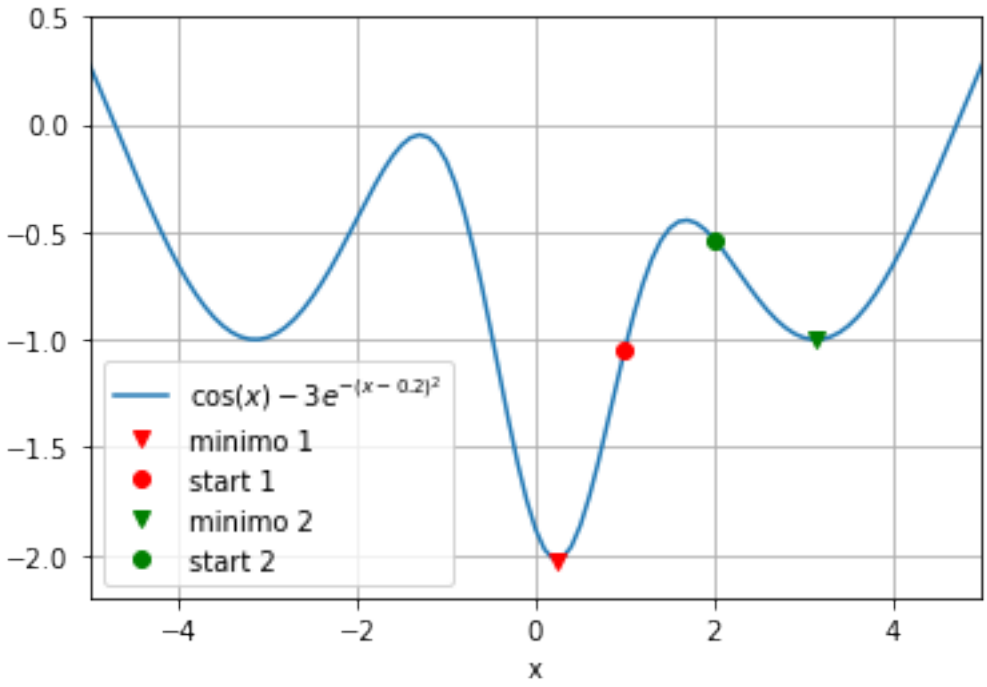
A seguir apresentamos um exemplo de como utilizar a rotina de otimização para minimizar $f(x)=\cos(x)-3\exp(-(x-0.2)2)$.

Chamamos o pacote de otimização digitando scipy.optimize.fmin que precisa de dois argumentos, o valor $x0$ a partir do qual iniciamos a busca para o mínimo.

In [186]:
```
from scipy import arange, cos, exp
from scipy.optimize import fmin
import pylab

def f(x):
    return cos(x) - 3 * exp( -(x - 0.2) ** 2)

# determina  minimo de f(x),
# inicia a partir de 1.0 e de 2.0, respectivamente
minimum1 = fmin(f, 1.0)
print("Inicia a pesquisa com  x=1., minimo é", minimum1)
minimum2 = fmin(f, 2.0)
print("Inicia a pesquisa com x=2., minimo é", minimum2)

# plota função
x = arange(-10, 10, 0.1)
y = f(x)
pylab.plot(x, y, label='$\cos(x)-3e^{-(x-0.2)^2}$')
pylab.xlabel('x')
pylab.grid()
pylab.axis([-5, 5, -2.2, 0.5])

# adiciona o minimo ao plot
pylab.plot(minimum1, f(minimum1), 'vr',
           label='minimo 1')
# adiciona o ponto inicial ao plot
pylab.plot(1.0, f(1.0), 'or', label='start 1')

# adiciona o minimo ao plot
pylab.plot(minimum2,f(minimum2),'vg',\
           label='minimo 2')
# adiciona o ponto segundo inicial ao plot
pylab.plot(2.0,f(2.0),'og',label='start 2')

pylab.legend(loc='lower left')
```

```
Optimization terminated successfully.
         Current function value: -2.023866
         Iterations: 16
         Function evaluations: 32
Inicia a pesquisa com  x=1., minimo é [0.23964844]
Optimization terminated successfully.
         Current function value: -1.000529
         Iterations: 16
         Function evaluations: 32
Inicia a pesquisa com x=2., minimo é [3.13847656]
```

Out[186]: <matplotlib.legend.Legend at 0x1fb43f20ef0>

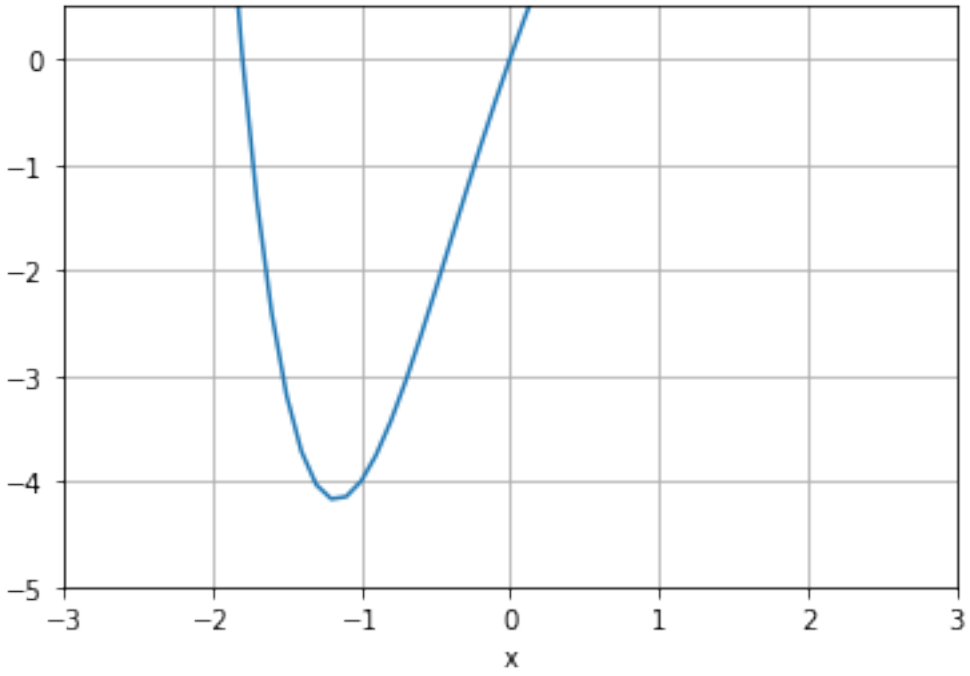



```
In [187]: from scipy import arange, cos, exp
          from scipy.optimize import fmin
          import pylab

          def f(x):
              return x**4-x**2+4*x

          # determina  minimo de f(x),
          # inicia a partir de -1.0 e de 1.0, respectivamente
          minimum1 = fmin(f, -1.0)
          print("Inicia a pesquisa com  x=-1., minimo é", minimum1)
          minimum2 = fmin(f, 1.0)
          print("Inicia a pesquisa com x=1., minimo é", minimum2)

          # plota função
          x = arange(-3, 3, 0.1)
          y = f(x)
          pylab.plot(x, y, label='$teste$')
          pylab.xlabel('x')
          pylab.grid()
          pylab.axis([-3, 3, -5, 0.5])

Optimization terminated successfully.
         Current function value: -4.175166
         Iterations: 12
```

```
         Function evaluations: 24
Inicia a pesquisa com  x=-1., minimo é [-1.16533203]
Optimization terminated successfully.
         Current function value: -4.175166
         Iterations: 19
         Function evaluations: 38
Inicia a pesquisa com x=1., minimo é [-1.16533203]
```

Out[187]:

$$(-3.0, \quad 3.0, \quad -5.0, \quad 0.5)$$



14.0.1 Epsilon da Máquina

In [188]:
```
eps = 1.0
while 1.0 + eps > 1.0:
    eps = eps / 2.0
print(eps)
```

```
1.1102230246251565e-16
```

15 Aprendendo a Programar: exemplos

15.0.1 Exemplo 1: usando o comando for

```
In [189]: for i in range(10):
              print(2**i - 1)
```

```
0
1
3
7
15
31
63
127
255
511
```

15.0.2 Exemplo 2: usando o comando while

```
In [190]:  eps = 1.0
           while 1.0 + eps > 1.0:
               eps = eps / 2.0
           print(eps)
```

```
1.1102230246251565e-16
```

16 Função anônima

Python tem uma infinidade de funções matemáticas já definidas tais como ceil, floor, fabs, factorial, log, exp, sin, cos, etc. Mas as vezes precisamos de outras.

Vejamos como definir uma função matemática.

As funções anônimas — em Python também chamadas de expressões lambda — representam um recurso bem interessante da linguagem Python, mas cuja utilidade pode não ser muito óbvia à primeira vista.

Uma função anônima é útil principalmente nos casos em que precisamos de uma função para ser passada como parâmetro para outra função, e que não será mais necessária após isso, como se fosse "descartável".

Vamos definir uma função de dois modos diferentes: usando o def e usando lambda.

```
In [191]: def func1(x):
              return x**2
```

```
In [192]: func1(2)
```

```
Out[192]:
```

```
In [193]: func2 =  lambda x: x**2

In [194]: func2(5)

Out[194]:
```

25

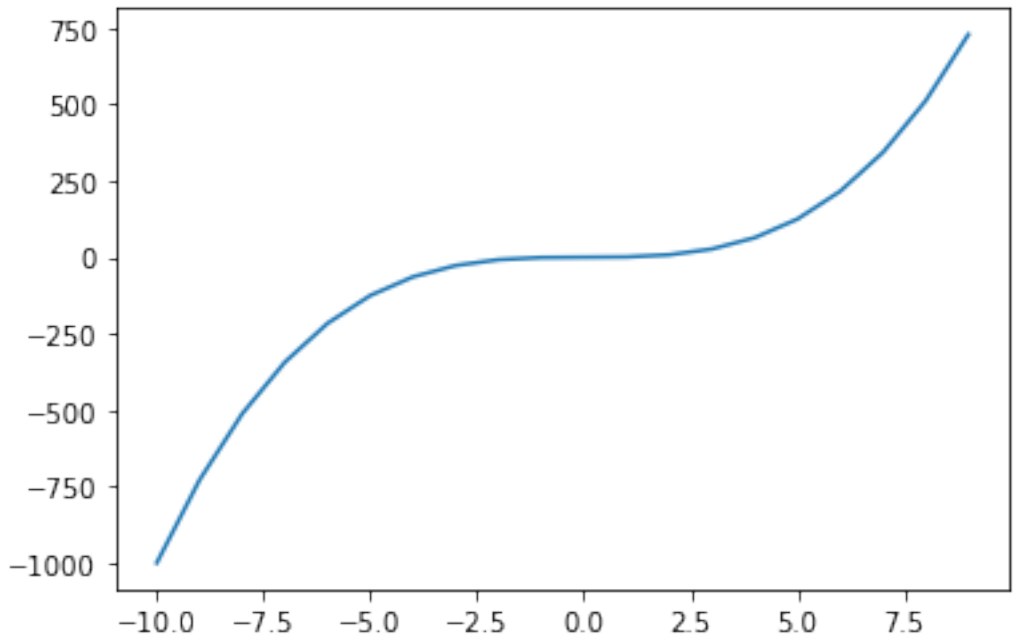
```
In [195]: func2(math.pi)

Out[195]:
```

9.869604401089358

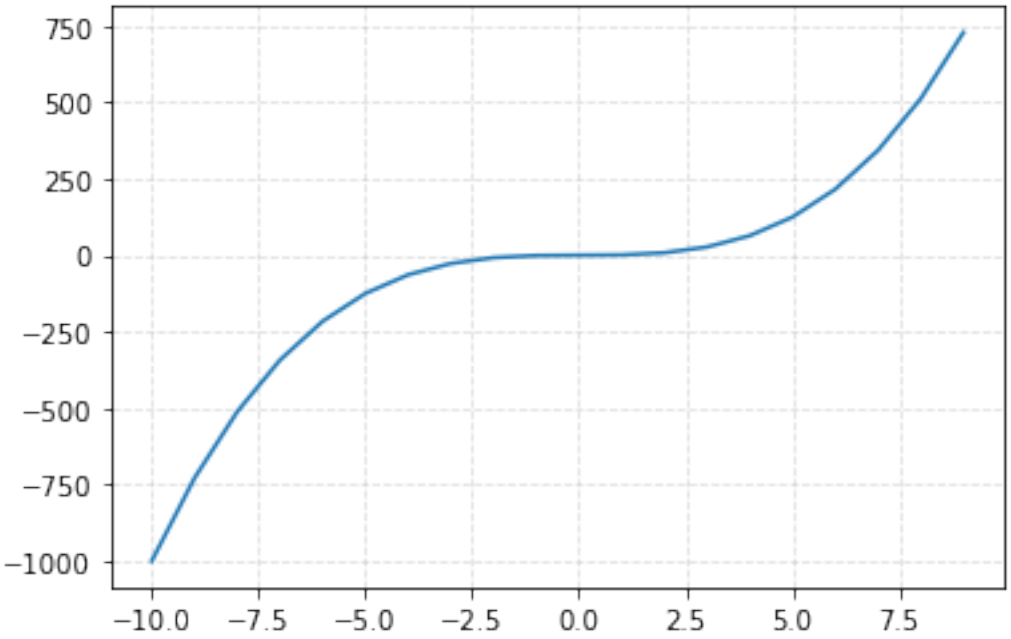
```
In [ ]:

In [196]: #  log de base e
          print ("Logaritmo Natural de 14 é : ", end="")
          print (math.log(14))

Logaritmo Natural de 14 é : 2.6390573296152584


In [197]: # log de 14 na base 5
          print ("Logaritmo de 14 na base 5 é : ", end="")
          print (math.log(14,5))

Logaritmo de 14 na base 5 é : 1.6397385131955606
```

Plotando o gráfico de uma função

```
In [198]: # Importa módulos que serão necessários
          import matplotlib.pyplot as plt
          import numpy as np

          # Cria vetores  X e Y
          x = np.array(range(-10,10))
          y = x ** 3

          # Cria o plot
          plt.plot(x,y)

          # Mostra o plot
          plt.show()
```



In [199]:
```
# Importa módulos que serão necessários
import matplotlib.pyplot as plt
import numpy as np

# Cria vetores  X e Y
x = np.array(range(-10,10))
y = x ** 3

# Cria um grid para melhor visualização
plt.grid(alpha=.4,linestyle='--')

# Cria o plot
plt.plot(x,y)

# Mostra o plor
plt.show()
```



17 Função Zeta de Riemann

A função Zeta de Riemann é definida pela soma infinita (série):

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}.$$

Vamos escrever um programinha que calcula o valor $\zeta(n)$ para um N fornecido:

$$\sum_{n=1}^{N} \frac{1}{n^s}$$

```
In [200]: def zeta(s,N):
              "Calcula a N soma parcial da função Zeta de Riemann em s."
              terms = [1/n**s for n in range(1,N+1)]
              partial_sum = sum(terms)
              return partial_sum

In [201]: zeta(2,10)

Out[201]:
```

1.5497677311665408

Alguns valores especiais da função Zeta de Riemann:

$$\zeta(2) = \frac{\pi^2}{6} \qquad \text{and} \qquad \zeta(4) = \frac{\pi^4}{90}$$

Vamos usar a função zeta acima para calcular uma aproximação de $\zeta(2)$, com $s = 2$ e $N = 100000$:

```
In [202]: zeta(2,100000)
```

```
Out[202]:
```

1.644924066898227

```
In [203]: zeta(4,100000)
```

```
Out[203]:
```

1.0823232337111381

18 Area de triangulo: fórmula de Heron

A fórmula de Heron calcula a área de um triângulo utilizando o semi-perímetro s do triângulo. Lembrando que o semi-perímetro de um triängulo de lados com medidas a, b e c é dado por

$$s = \frac{a+b+c}{2}.$$

A área do triangulo pela fórmula de Heron é:

$$A = \sqrt{s(s-a)(s-b)(s-c)}.$$

Vamos criar uma função que calcula a área de um triângulo utilizando a fórmula de Heron conhecido os vértices do triângulo.

```
In [204]: def area_triangulo(vertices):
              '''Calcula a área conhecendo os vértices.

              Parametros
              ----------
              vertices : os vertices do triangulo [(x1,y1),(x2,y2),(x3,y3)].

              Retorna
              -------
              Area do triângulo pela fórmula de Heron.

              Examplo
              --------
              area_triangulo([(-1,2),(-3,-1),(4,1)])
               '''
              # calculando distancia do vertice 1 ao vertice2
              # calculando (x_1-x_2)
              a_x = abs(vertices[0][0] - vertices[1][0])
              # calculando (y_1-y_2)
              a_y = abs(vertices[0][1] - vertices[1][1])
              # calculando distancia do vertice 1 ao vertice2
              a = (a_x**2 + a_y**2)**0.5
```

```
        # calculando a distancia entre os vertices 1 e 2
        b_x = abs(vertices[1][0] - vertices[2][0])
        b_y = abs(vertices[1][1] - vertices[2][1])
        b = (b_x**2 + b_y**2)**0.5

        # calculando a distancia entre os vertices 0 e 2
        c_x = abs(vertices[0][0] - vertices[2][0])
        c_y = abs(vertices[0][1] - vertices[2][1])
        c = (c_x**2 + c_y**2)**0.5

        # calculando o semiperimentro
        s = (a + b + c)/2
        # Calculando a área usando a formula de Heron
        area = (s*(s - a)*(s - b)*(s - c))**0.5

        return area
```

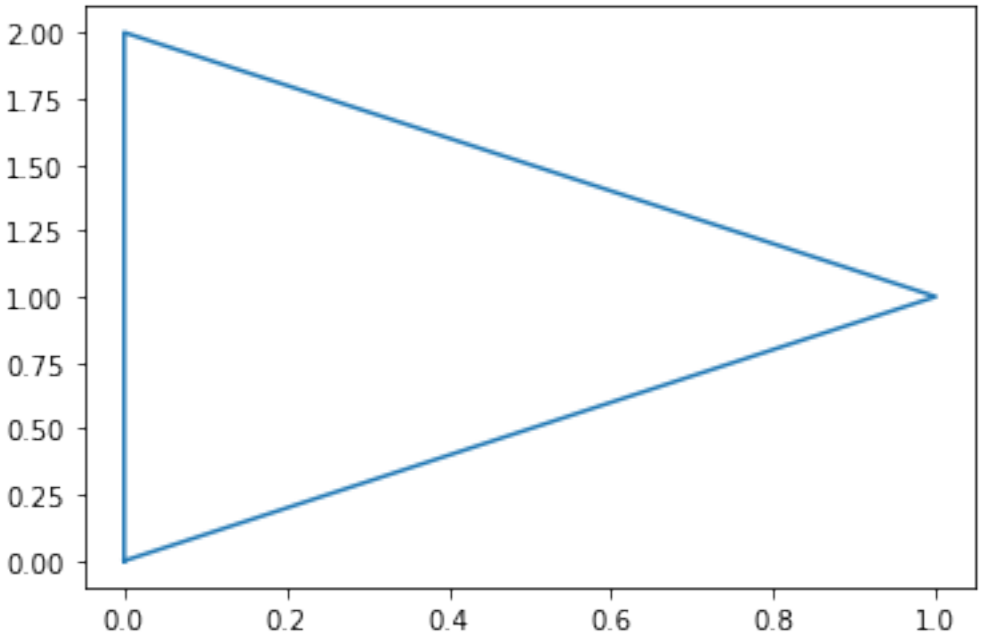
In [205]:
```
area_triangulo([(0,0),(0,1),(1,0)])
```

Out[205]:

0.49999999999999983

Vejamos o desenho deste triangulo.

In [206]:
```
import matplotlib.pyplot as plt

coord = [[0,0], [0,2], [1,1]]
coord.append(coord[0]) #repete o primerio criando o poligono'

xs, ys = zip(*coord) #crea uma lista de x e de y

plt.figure()
plt.plot(xs,ys)
plt.show()
```




```
In [207]: area_triangulo([(-1,2),(-3,-1),(4,1)])
```

Out[207]:

8.499999999999996

```
In [208]: area_triangulo([(0,0),(0,2),(1,1)])
```

Out[208]:

0.9999999999999997

```
In [ ]:
```