AULA 5

Raízes de equações não lineares.

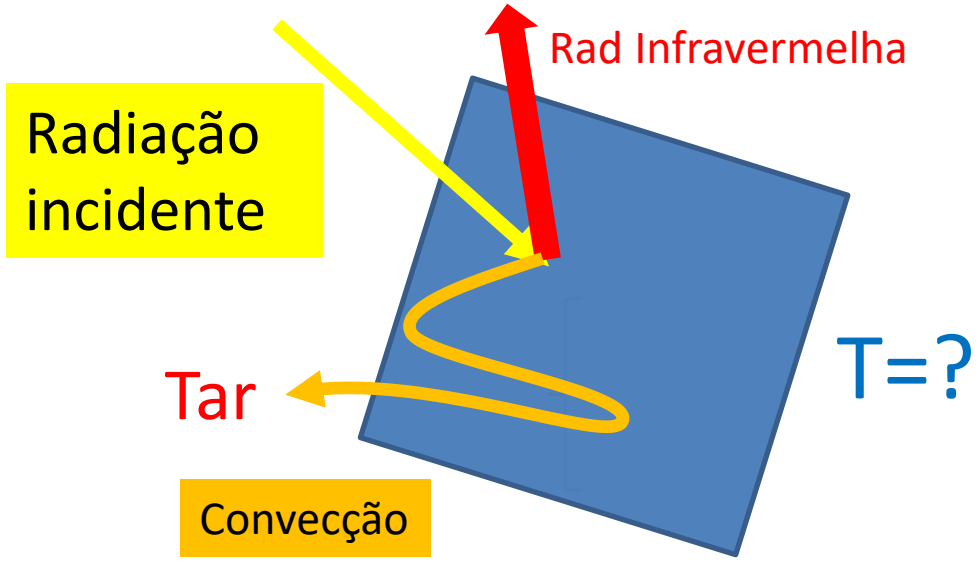
Equilíbrio térmico de um painel solar.

Método da bisseção.

Equilíbrio térmico de um painel solar

Vamos estudar um exemplo de termodinâmica: o equilíbrio térmico de um painel solar.

Determinar a temperatura de equilíbrio de uma placa negra exposta ao sol e ao ar.



Balanço energético

1º Princípio da termodinâmica (conservação da energia):

Fluxo de calor recebido - Fluxo de calor perdido=0

A placa perde calor sob duas formas: por radiação e por condução/convecção.

Arrefecimento por Radiação: lei de Stefan-Boltzmann:

$$\frac{dQ}{dt} = -\sigma T^4 \ [Wm^{-2}]$$

Constante de Stefan-Boltzmann:

$$\sigma = 5.67 \times 10^{-8} \ Wm^{-2}K^{-4}$$

Arrefecimento por condução/convecção

O calor transferido para o ar, por condução/convecção (lei de Newton):

$$\frac{dQ}{dt} = -\alpha\,(T - T_0)$$

α é uma constante numérica, e.g. $\alpha = 0.4\ Wm^{-2}K^{-1}$

Equação de balanço térmico

$$\frac{dQ}{dt} = E_{inc} - \sigma T^4 - \alpha\,(T - T_0) = 0$$

Trata-se de uma equação não linear com uma incógnita (T).

Tratando-se de um polinómio de 4ª ordem, existem 4 raízes, e até existe fórmula resolvente. Só interessa uma raiz que seja real e positiva.

Parâmetros: $E_{inc} = 500 W m^{-2}, \alpha = 0.4\, W m^{-2} K^{-1}, T_0 = 273 K$

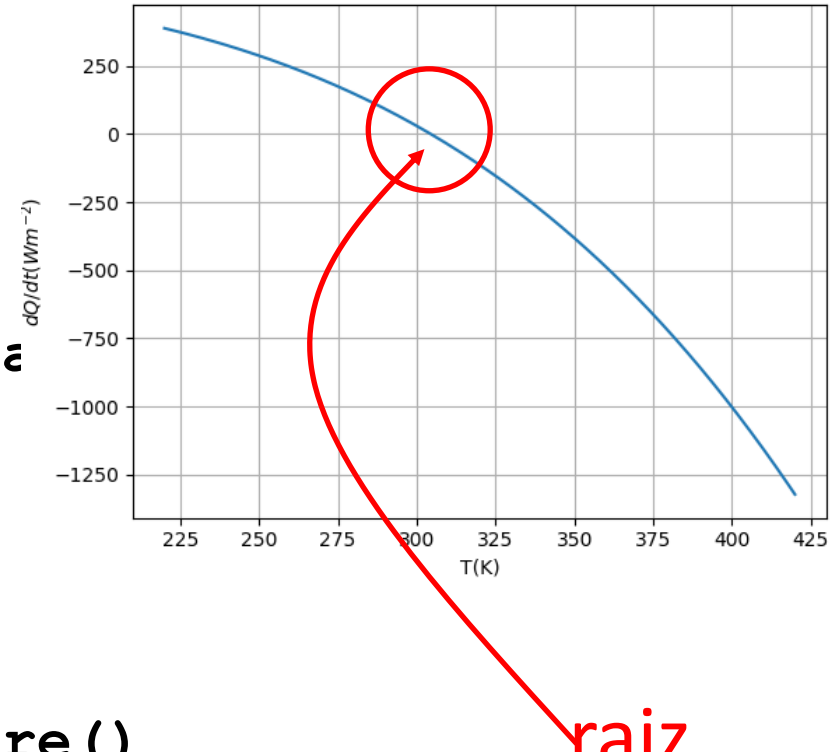
Solução exata

```
from sympy import Symbol,solve
T=Symbol('T')
T0=273;sigma=5.67e-8;alpha=0.4;Es=500;
Teq=solve(Es-sigma*T**4-alpha*(T-T0),T)
print(Teq)
```

```
>> [-338.521492377835,
304.492292028057,
17.0146001748888 - 322.406071549869*I,
17.0146001748888 + 322.406071549869*I]
```

4 raizes

Solução gráfica




```
import numpy as np
import matplotlib.pyplot a
def y(T):
    f=500-5.67e-8*T**4\
      -0.4*(T-273)
    return f
plt.close('all');plt.figure()
T=np.linspace(220,420,201)
plt.plot(T,y(T))
plt.xlabel('T(K)')
plt.ylabel(r'${dQ}/{dt}  (Wm^{-2})$')
plt.grid()
```

Determinação de raízes de equações não lineares: caso geral (1 dimensão)

Calcular

$$x: \quad f(x) = 0$$

Com a condição $x \in \mathbb{R}$, ou mais restrita $x \in \mathbb{R}^+$

Salvo casos particulares, as equações não lineares não podem ser resolvidas analiticamente; i.e. não podem ser resolvidas explicitamente em ordem a x.

Podemos desenhar métodos numéricos iterativos, que partem duma estimativa inicial do valor da raiz, procedendo por iterações sucessivas.

Dois métodos: Método da bissecção e Método de Newton-Raphson

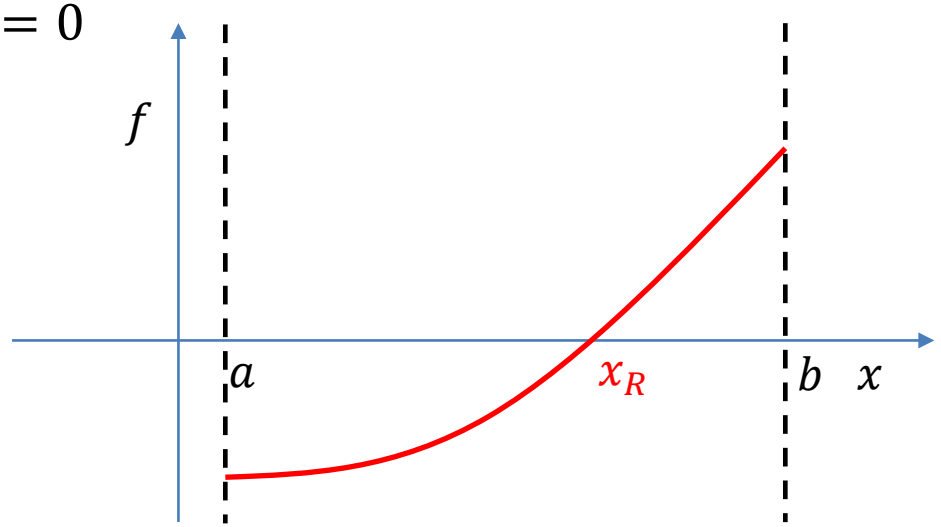
Método da bissecção (1 dimensão)

Baseia-se no teorema do valor médio:

Se $f(x)$ é contínua no intervalo $[a, b]$ e $f(a)f(b) < 0$,

então $\exists x_R \in [a, b]: f(x_R) = 0$

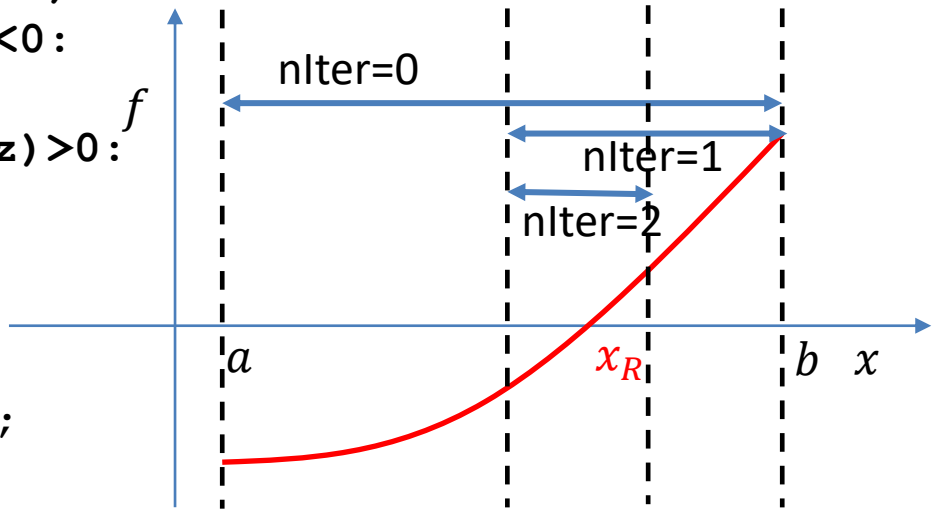
x_R é a raiz procurada



O método consiste na divisão sucessiva do intervalo original em duas partes iguais seguida da escolha do subintervalo em que ocorre a mudança do sinal.

Método da bisseção calcular R: $y(R) = 0$

```
def bissec(y,xLow,xHigh,maxErro,maxIter):
    nIter=0
    if y(xLow)*y(xHigh)>0:
        raiz=float('nan')  # not a number: ERRO
    else:
        erroAbs=xHigh-xLow
        while erroAbs>maxErro and nIter<maxIter:
            nIter=nIter+1
            raiz=0.5*(xHigh+xLow)
            if y(xLow)*y(raiz)<0:
                xHigh=raiz
            elif y(xLow)*y(raiz)>0:
                xLow=raiz
            else:
                xHigh=raiz
                xLow=raiz
            erroAbs=xHigh-xLow;
return raiz,erroAbs,nIter
```



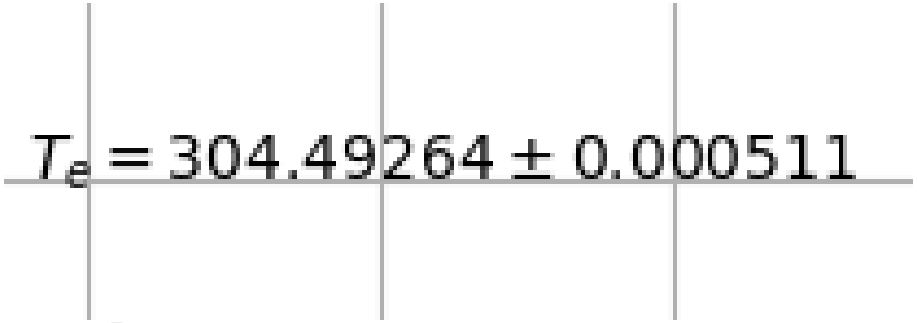
Controlo da convergência

```
while erroAbs>maxErro and nIter<maxIter:
```

Condição de convergência

Válvula de seguranca
(Evita ciclos infinitos)

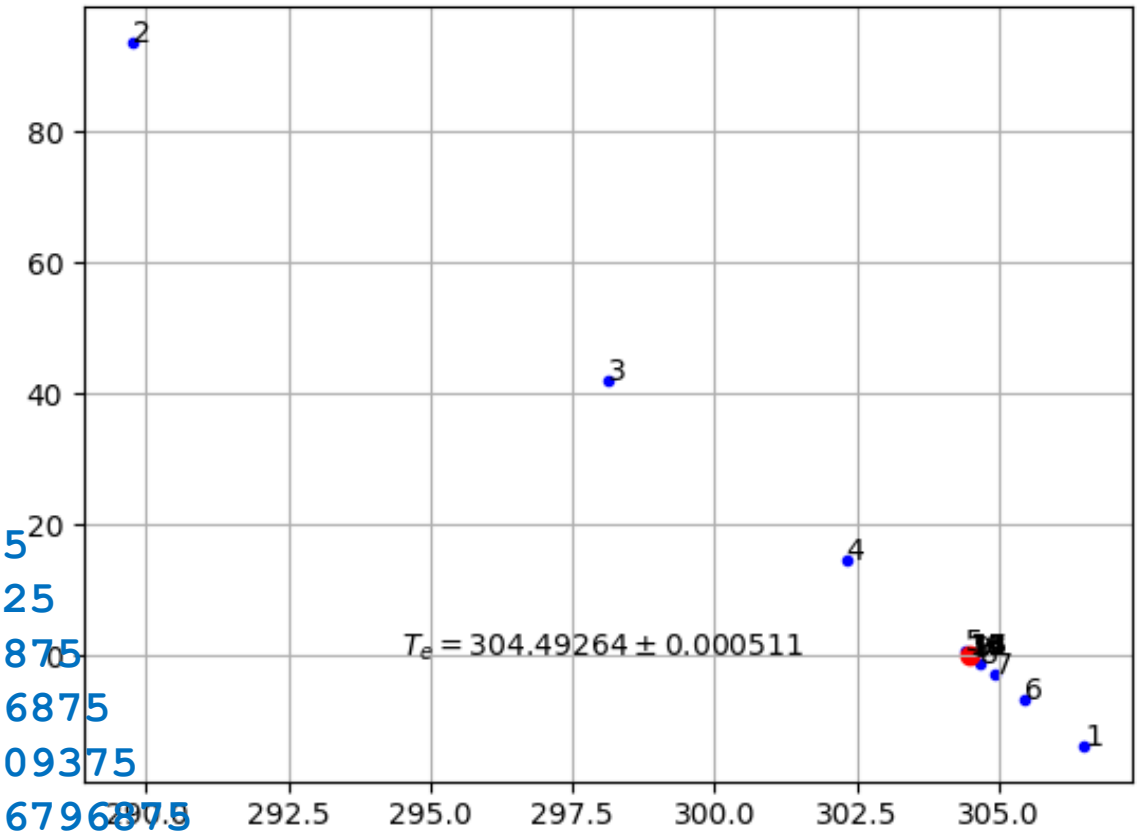
Aplicação




```
import matplotlib.pyplot as plt
import numpy as np
plt.close('all')
TLow=273; THigh=340; #intervalo inicial (K)
maxIter=100; #numero maximo de iteracoes
maxErro=0.001; #erro absoluto máximo (K)
Teq, erroAbs,
nIter=bissec(y,TLow,THigh,maxErro,maxIter)
plt.scatter(Teq,y(Teq),marker='o',color='r')
plt.text(Teq-10,y(Teq),\
        r'$T_{e}=%10.5f \pm %8.6f$' % (Teq,erroAbs))
plt.grid()
```

output

```
nIter= 1 raiz= 306.5
nIter= 2 raiz= 289.75
nIter= 3 raiz= 298.125
nIter= 4 raiz= 302.3125
nIter= 5 raiz= 304.40625
nIter= 6 raiz= 305.453125
nIter= 7 raiz= 304.9296875
nIter= 8 raiz= 304.6679687 5
nIter= 9 raiz= 304.537109375
nIter= 10 raiz= 304.4716796875
nIter= 11 raiz= 304.50439453125
nIter= 12 raiz= 304.488037109375
nIter= 13 raiz= 304.4962158203125
nIter= 14 raiz= 304.49212646484375
nIter= 15 raiz= 304.4941711425781
nIter= 16 raiz= 304.49314880371094
nIter= 17 raiz= 304.492 63763427734
```



Exato= 304.492292028057

Caso maxErro=10^{-14}

nIter= 1 raiz= 306.5
nIter= 2 raiz= 289.75
…
nIter= 17 raiz= 304.49263763427734
nIter= 23 raiz= 304.49229419231415
nIter= 25 raiz= 304.49229219555855
nIter= 29 raiz= 304.4922920707613
nIter= 34 raiz= 304.492292078623
nIter= 36 raiz= 304.4922920283726
nIter= 40 raiz= 304.492292028451
nIter= 43 raiz= 304.4922920280527
nIter= 47 raiz= 304.492292028057
…
nIter= 99 raiz= 304.4922920280569
nIter= 100 raiz= 304.4922920280569

Exato= 304.492292028057

Não é possível esse erro absoluto em float64.

Em cada iteração só se pode ganhar 1 bit:
$\pm$ 1 algarismo/3-4 iterações

Solução numérica python

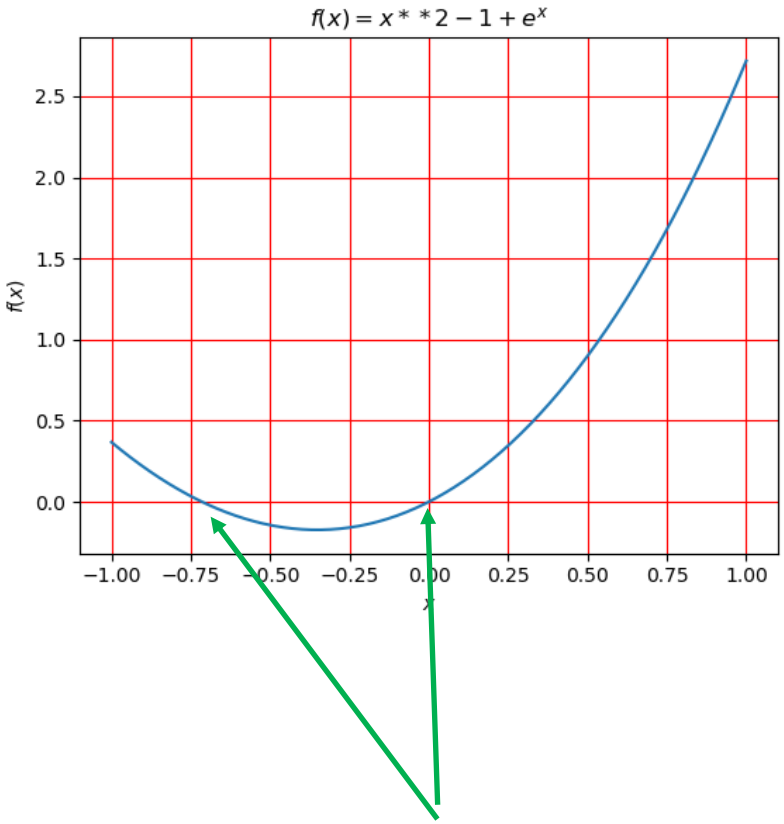
```
from scipy.optimize import fsolve
#300 é uma estimativa à priori da solução
Teq2=fsolve(y,300)
print(Teq2)
```

```
>>[ 304.49229203]
```
Exato= 304.492292028057

Raiz da equação $f(x) = x^2 - 1 + e^x$

Não tem solução analítica
A bisseção funcionaria se o intervalo de partida contivesse (só) 1 raiz

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    y=x**2-1+np.exp(x)
    return y
plt.figure()
xis=np.linspace(-1,1,201)
plt.plot(xis,f(xis))
plt.grid(color='red')
plt.xlabel(r'$x$')
plt.title(r'$f(x)=x**2-1+e^{x}$')
plt.ylabel(r'$f(x)$')
```



Raizes $\in [-1,1]$

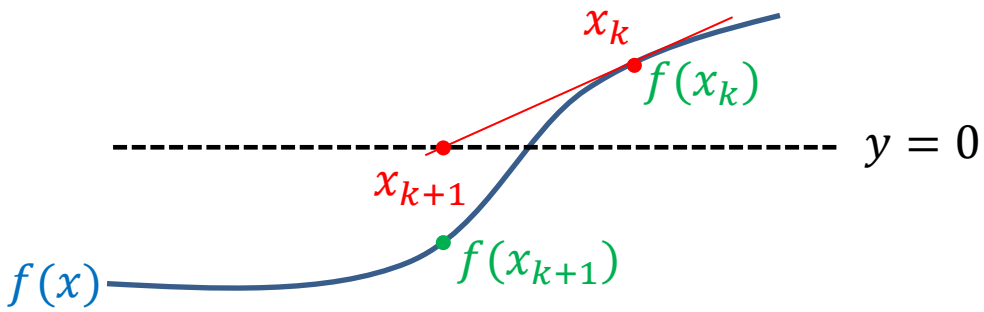
Método de Newton (-Raphson)

No método da bissecção a única informação utilizada é o sinal da função f nos extremos dos subintervalos.

Se f for diferenciável, é possível construir um método mais eficiente para encontrar os zeros da função f.

O método de Newton-Raphson baseia-se diretamente no desenvolvimento em série de Taylor da função f em torno de um ponto nas proximidades de uma raiz.

Série de Taylor



$$f(x_{k+1}) = f(x_k) + f'(x_k)(x_{k+1} - x_k) + \cdots + \frac{1}{n!} f^n(x_k)(x_{k+1} - x_k)^n$$

Onde

$$f'(x_k) = \left(\frac{df}{dx}\right)_{x=x_k}, f^n(x_k) = \left(\frac{d^n f}{dx^n}\right)_{x=x_k}$$

Impondo a condição

$$f(x_{k+1}) = 0$$

e desprezando termos de ordem superior à primeira, obtém-se:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

desde que $f'(x_k) \neq 0$.

Recursão: $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

A equação acima pode então ser utilizada de forma recursiva para, a partir de uma estimativa inicial do valor da raiz x_0, obter aproximações sucessivamente melhores do seu valor:

$$x_0, x_1, x_2, \ldots, x_N$$

até um número máximo de iterações (N) ou ser atingido um critério de convergência.

A convergência pode ser bastante rápida (quadrática).

Mas, contrariamente ao método da bissecção, não existe garantia de convergência do método de Newton.

Método de Newton

A fórmula recursiva permite construir uma sucessão x[k] a partir de um valor inicial x[0].

O método de Newton consiste em substituir localmente a função f pela sua tangente…

Vejamos o seguinte exemplo:

$$f(x) = x^2 - 1 + e^x$$

Partindo do dado inicial: x[0]=0.5

Com uma tolerância de 10^{-6}

Cálculo da derivada usando sympy

```
from sympy import Symbol,diff,exp
x=Symbol('x')
fp=diff(x**2-1+exp(x),x)
print(fp)
>>2*x + exp(x)
```

$$x\text{: } f(x) = x^2 - 1 + e^x = 0$$

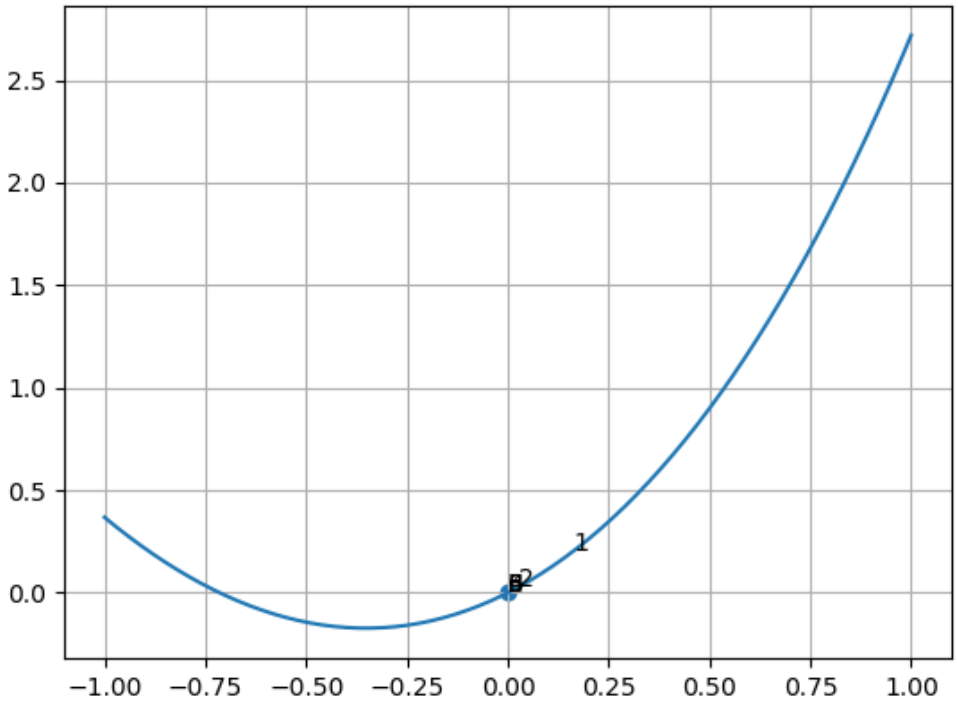
```
import numpy as np
def f(x):
    fun=x**2-1+np.exp(x)
    return fun
def fprime(x):
    fp=2*x+np.exp(x)
    return fp
```

…

$$x\text{: } f(x) = x^2 - 1 + e^x = 0$$

```
…
def newt(fx,dfdx,xguess,maxerr):
    maxiter=100
    xold=xguess
    erro=10*maxerr
    xnew=xguess+erro
    niter=0
    while erro>maxerr and niter<maxiter:
        niter=niter+1
        xnew=xold-fx(xold)/dfdx(xold)
        erro=abs(xold-xnew)
        xold=xnew
    return xnew,erro,niter
raiz,err,ni=newt(f,fprime,0.5,0.1)
print(raiz,err,ni)
```

Convergência
1as iterações



Convergiu para uma das raízes

Raiz=4.423146143015834e-17 Erro=3.4736657994109905e-12 ITER=6