Code Sessions para Métodos Numéricos

Projeto Numbiosis

Gustavo Charles P. de Oliveira
Departamento de Computação Científica
Universidade Federal da Paraíba

13 de Abril de 2020

Conteúdo

1 Code session 1

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Determinação de raízes

`bisect`
A função `bisect` localiza a raiz de uma função dentro de um intervalo dado usando o método da bisseção. Os argumentos de entrada obrigatórios desta função são:

1. a função-alvo `f` (contínua)
2. o limite esquerdo `a`
3. o limite direito `b`

Parâmetros opcionais relevantes são:

- `xtol`: tolerância (padrão: 2e-12)
- `maxiter`: número máximo de iterações (padrão: 100)
- `disp`: mostra erro se algoritmo não convergir (padrão: True)

O argumento de saída é:

- `x0`: a estimativa para a raiz de `f`

Como importá-la?

```
from scipy.optimize import bisect
```

[2]:
```
from scipy.optimize import bisect
```

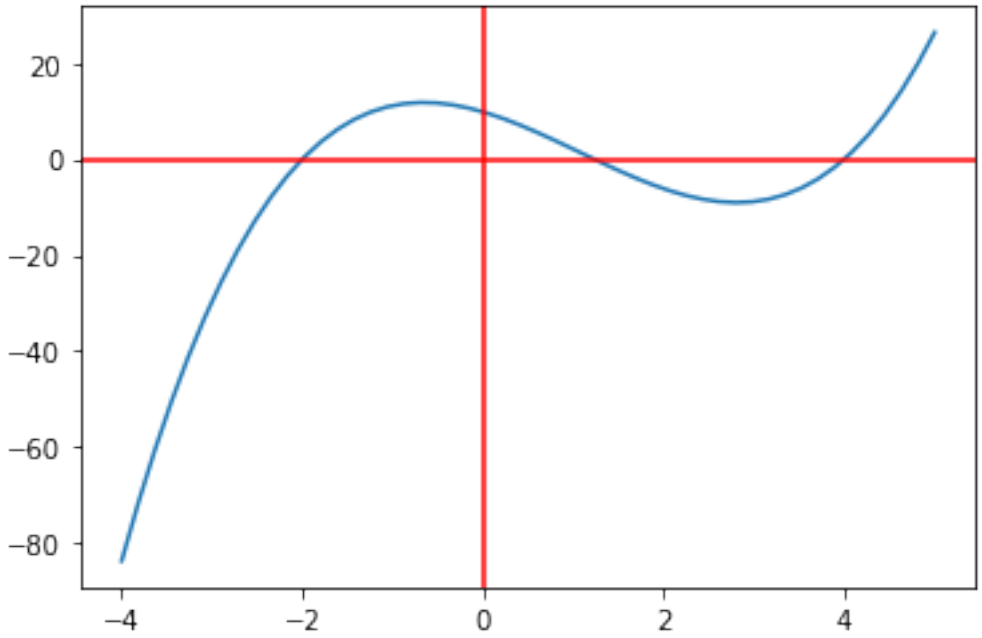
1.1.1 Problema 1

Encontre a menor raiz positiva (real) de $x^3 - 3.23x^2 - 5.54x + 9.84 = 0$ pelo método da bisseção.

Resolução

[3]:
```
# função
def f(x):
    return x**3 - 3.23*x**2 - 5.54*x + 9.84
```

[4]:
```
# analise gráfica
x = np.linspace(-4,5)
plt.plot(x,f(x));
plt.axhline(y=0,color='r');
plt.axvline(x=0,color='r');
```



Pelo gráfico, vemos que a menor raiz positiva está localizada no intervalo $(0,2]$. Vamos determiná-la utilizando este intervalo de confinamento.

[5]:
```
# resolução com bisect

x = bisect(f,0,2) # raiz

print('Raiz: x = %f' % x)
```

```
Raiz: x = 1.230000
```

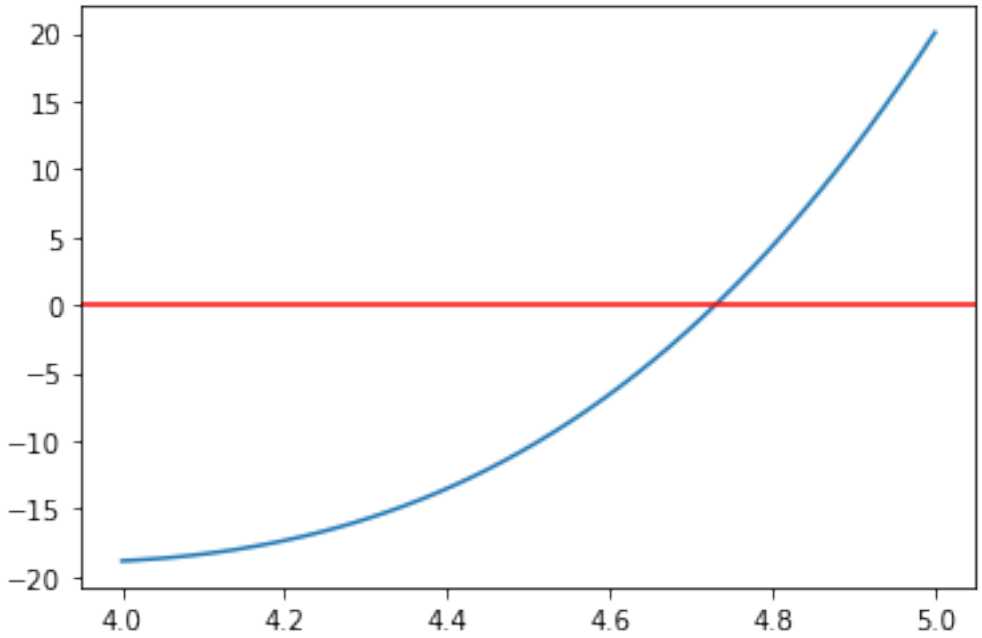
1.1.2 Problema 2

Determine a menor raiz não nula positiva de $\cosh(x)\cos(x) - 1 = 0$ dentro do intervalo $(4,5)$.

Resolução Sigamos o procedimento aprendido com `bisect`.

[6]:
```
# função
def f(x):
    return np.cosh(x)*np.cos(x) - 1
```

[7]:
```
# analise gráfica
x = np.linspace(4,5)
plt.plot(x,f(x));
plt.axhline(y=0,color='r');
```



[8]:
```
# resolução com bisect

x = bisect(f,4,5) # raiz

print('Raiz: x = %f' % x)
```

```
Raiz: x = 4.730041
```

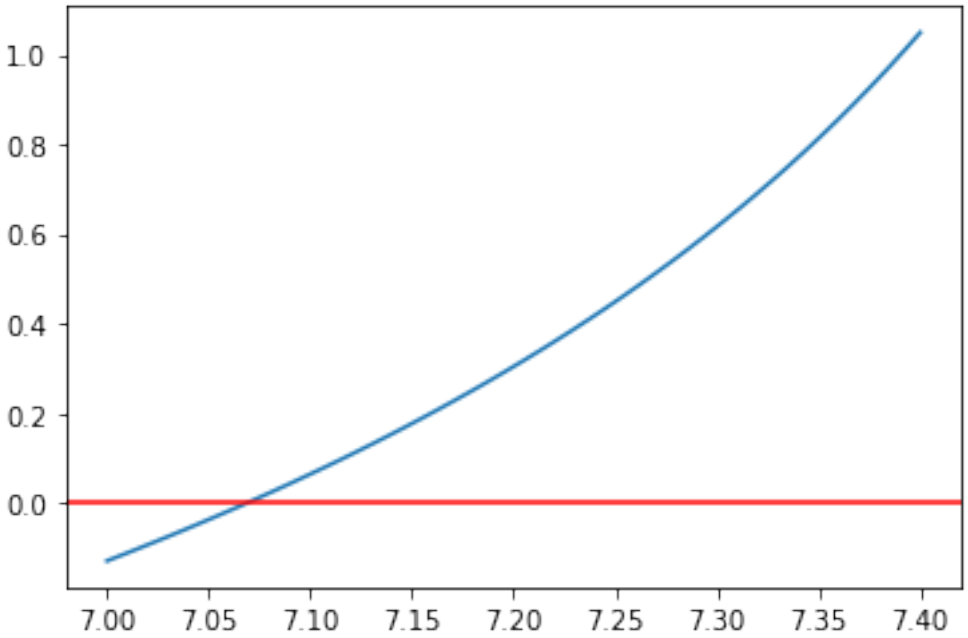
1.1.3 Problema 3

Uma raiz da equação $\tan(x) - \tanh(x) = 0$ encontra-se em $(7.0, 7.4)$. Determine esta raiz com três casas decimais de precisão pelo método da bisseção.

Resolução

[9]:
```
# função
def f(x):
    return np.tan(x) - np.tanh(x)
```

[10]:
```
# analise gráfica
x = np.linspace(7,7.4)
plt.plot(x,f(x));
plt.axhline(y=0,color='r');
```



Para obter as 3 casas decimas, vamos imprimir o valor final com 3 casas decimais.

[11]:
```
x = bisect(f,7,7.4) # raiz

print('Raiz: x = %.3f' % x)
```

```
Raiz: x = 7.069
```

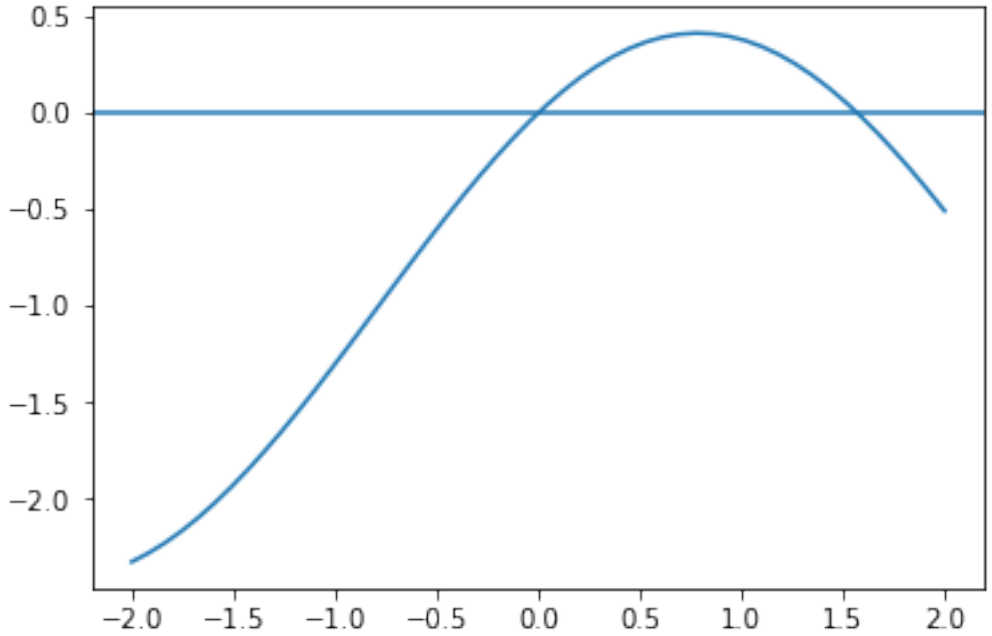
1.1.4 Problema 4

Determine as raízes de $\text{sen}(x) + 3\cos(x) - 1 = 0$ no intervalo $(-2, 2)$.

Resolução

[12]:
```
# função
def f(x):
    return np.sin(x) + np.cos(x) - 1
```

[13]:
```
# analise gráfica
x = np.linspace(-2,2)
plt.plot(x,f(x));
plt.axhline(y=0);
```



A análise gráfica mostra duas raízes. Vamos encontrar uma de cada vez.

[14]:
```
# resolução com bisect

x1 = bisect(f,-2,1) # raiz 1
x2 = bisect(f,1,2) # raiz 2

print('Raízes: x1 = %f; x2 = %f' % (x1,x2))
```

```
Raízes: x1 = -0.000000; x2 = 1.570796
```

1.1.5 Problema 5

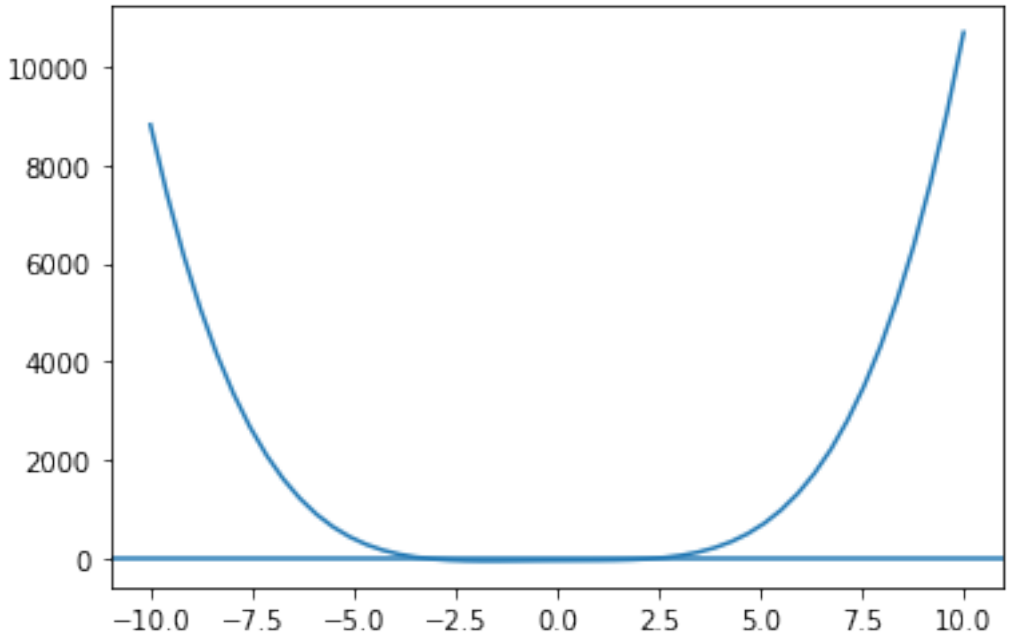
Determine todas as raízes reais de $P(x) = x^4 + 0.9x^3 - 2.3x^2 + 3.6x - 25.2$

Resolução

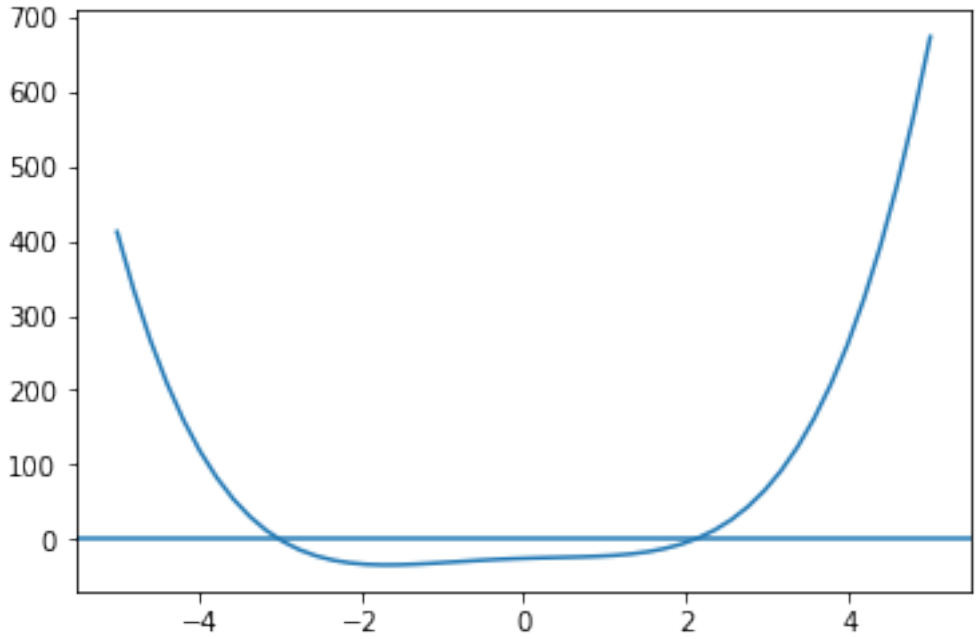
[15]:
```
# função
def P(x):
    return x**4 + 0.9*x**3 - 2.3*x**2 + 3.6*x - 25.2
```

[16]:
```
# define função para analise gráfica
def analise_grafica(xrange,f):
    plt.plot(xrange,f(xrange));
    plt.axhline(y=0);
```

[17]:
```
# analise gráfica 1
xrange = np.linspace(-10,10)
analise_grafica(xrange,P)
```



[18]:
```
# refinamento
xrange = np.linspace(-5,5)
analise_grafica(xrange,P)
```



[19]:
```
# resolução com bisect

x1 = bisect(P,-4,-2) # raiz 1
x2 = bisect(P,1,3) # raiz 2
```

```
print('Raízes: x1 = %f; x2 = %f' % (x1,x2))
```

```
Raízes: x1 = -3.000000; x2 = 2.100000
```

1.1.6 Problema 6

Um jogador de futebol americano está prestes a fazer um lançamento para outro jogador de seu time. O lançador tem uma altura de 1,82 m e o outro jogador está afastado de 18,2 m. A expressão que descreve o movimento da bola é a familiar equação da física que descreve o movimento de um projétil:

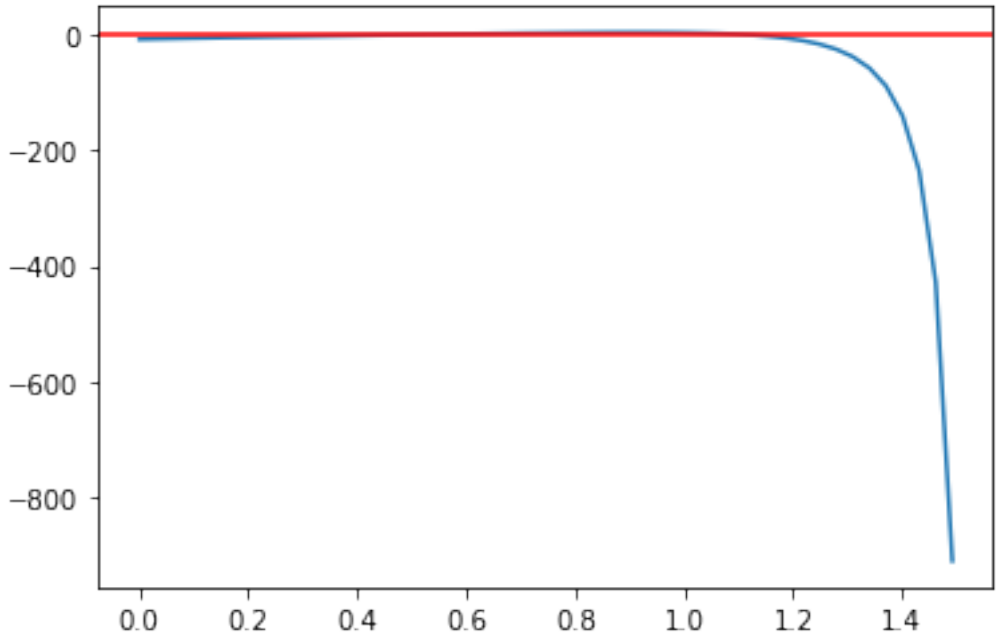
$$y = x\tan(\theta) - \frac{1}{2}\frac{x^2 g}{v_0^2}\frac{1}{\cos^2(\theta)} + h,$$

onde x e y são as distâncias horizontal e verical, respectivamente, $g = 9,8\,m/s^2$ é a aceleração da gravidade, v_0 é a velocidade inicial da bola quando deixa a mão do lançador e θ é o Ângulo que a bola faz com o eixo horizontal nesse mesmo instante. Para $v_0 = 15,2\,m/s$, $x = 18,2\,m$, $h = 1,82\,m$ e $y = 2,1\,m$, determine o ângulo θ no qual o jogador deve lançar a bola.

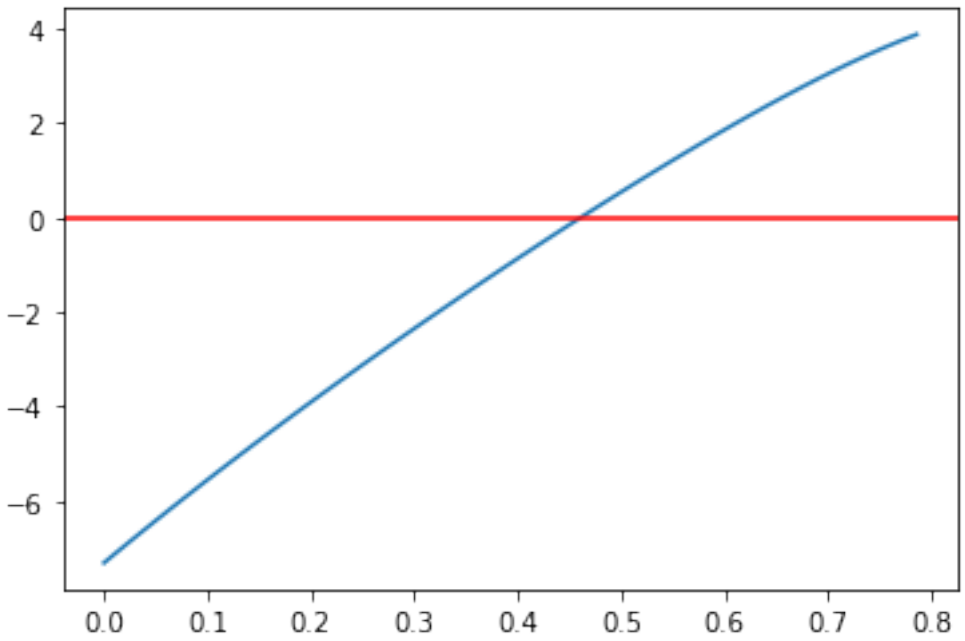
Resolução

[20]:
```
# parâmetros do problema
v0 = 15.2
x = 18.2
h = 1.82
y = 2.1
g = 9.8

# função f(theta) = 0
f = lambda theta: x*np.tan(theta) - 0.5*(x**2*g/v0**2)*(1/(np.cos(theta)**2)) + h␣
 ↪- y
```

[21]:
```
# análise gráfica
th = np.linspace(0,0.95*np.pi/2,50)
plt.plot(th,f(th));
plt.axhline(y=0,color='r');
```



[22]:
```
# análise gráfica - 2
th = np.linspace(0,np.pi/4,50)
plt.plot(th,f(th));
plt.axhline(y=0,color='r');
```



[23]:
```
# resolução por bisseção
xr = bisect(f,0.1,0.6)
print('Ângulo de lançamento: %.2f graus' % np.rad2deg(xr))
```

```
Ângulo de lançamento: 26.41 graus
```

1.1.7 Problema 7

A equação de Bernoulli para o escoamento de um fluido em um canal aberto com um pequeno ressalto é

$$\frac{Q^2}{2gb^2h_0^2} + h_0 = \frac{Q^2}{2gb^2h^2} + h + H,$$

onde $Q = 1.2\,m^3/s$ é a vazão volumétrica, $g = 9.81\,m/s^2$ é a aceleração gravitacional, $b = 1.8\,m$ a largura do canal, $h_0 = 0.6\,m$ o nível da água à montante, $H = 0.075\,m$ a altura do ressalto e h o nível da água acima do ressalto. Determine h.

Resolução Para este problema, definiremos duas funções, uma auxiliar, que chamaremos `a`, e a função `f(h)` que reescreve a equação de Bernoulli acima em função de h.

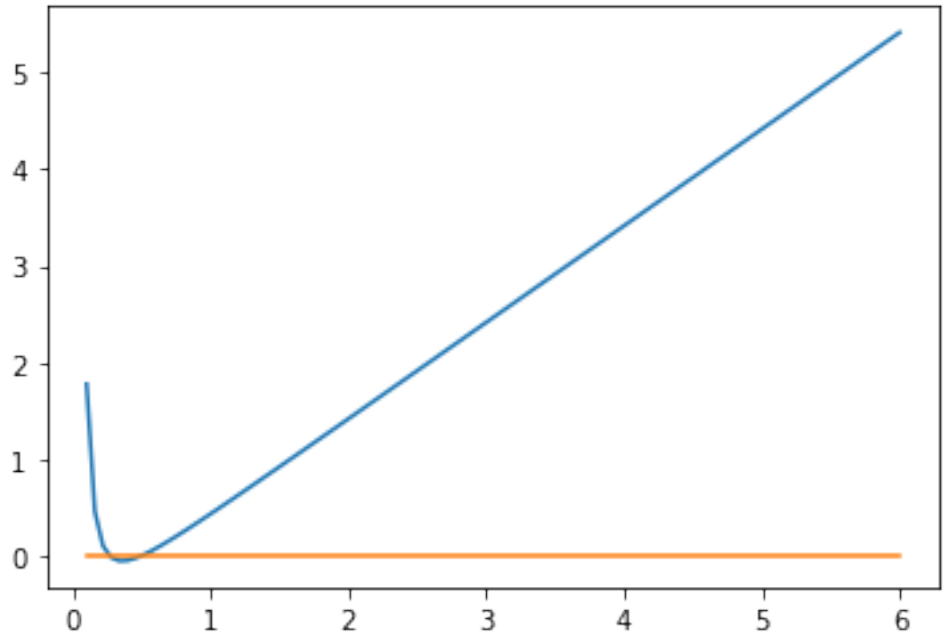
[24]:
```
# função para cálculo de parâmetros
def a(Q,g,b,H,h0):
    return Q,g,b,H,h0

# função do nível de água
def f(h):
    frate,grav,width,bHeight,ups = a(Q,g,b,H,h0)
    c = lambda arg: frate**2/(2*grav*width**2*arg**2)
    return c(h) - c(h0) + h - h0 + H
```

Note que a função `a` é apenas uma conveniência para o cálculo do termo comum envolvendo a vazão e para construírmos uma generalização para os dados de entrada. Em seguida, definiremos os parâmetros de entrada do problema.

[25]:
```
# parâmetros de entrada
Q = 1.2 # m3/s
g = 9.81 # m/s2
b = 1.8 # m
h0 = 0.6 # m
H = 0.075 # m
```

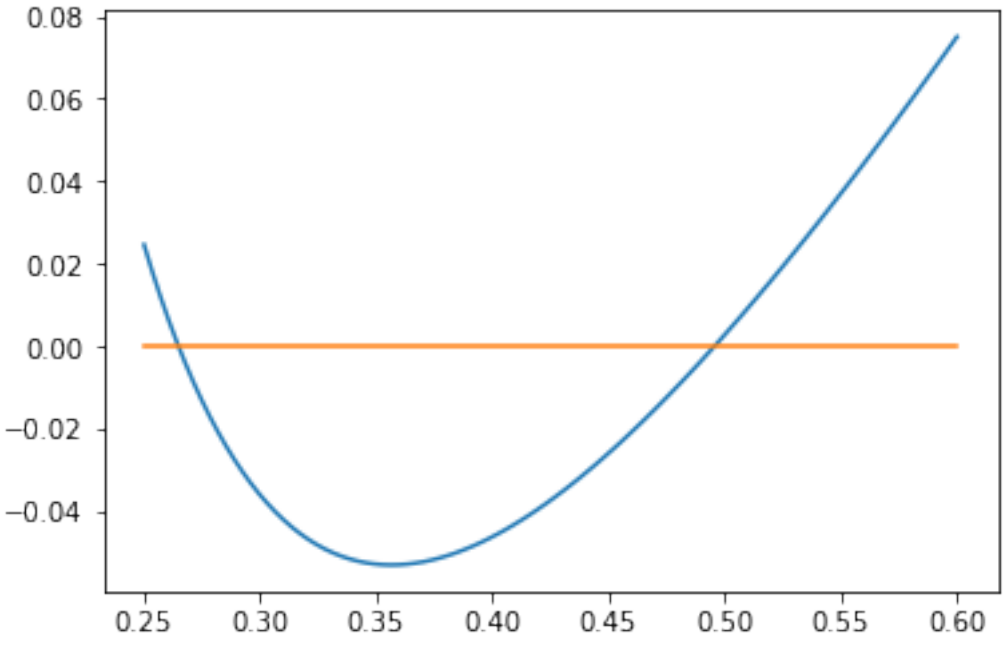
A partir daí, podemos realizar a análise gráfica para verificar o comportamento de `f(h)`.

[26]:
```
# análise gráfica
h = np.linspace(0.1,6,num=100)
plt.plot(h,f(h),h,f(h)*0);
```



Ampliemos a localização.

[27]:
```
# análise gráfica
h = np.linspace(0.25,0.6,num=100)
plt.plot(h,f(h),h,f(h)*0);
```



Verificamos que `f(h)` admite duas soluções. Vamos determinar cada uma delas.

[28]:
```
# solução
h1 = bisect(f,0.25,0.32)
print('Raiz: h1 = %f' % h1)
```

```
h2 = bisect(f,0.4,0.55)
print('Raiz: h2 = %f' % h2)
```

```
Raiz: h1 = 0.264755
Raiz: h2 = 0.495755
```

Nota: as duas soluções viáveis dizem respeito ao regime de escoamento no canal aberto. Enquanto h_1 é um limite para escoamento supercrítico (rápido), h_2 é um limite para escoamento subcrítico (lento).

1.1.8 Problema 8

A velocidade v de um foguete Saturn V em voo vertical próximo à superfície da Terra pode ser aproximada por

$$v = u\ln\left(\frac{M_0}{M_0 - \dot{m}t}\right) - gt,$$

onde $u = 2510\,m/s$ é a velocidade de escape relativa ao foguete, $M_0 = 2.8 \times 10^6\,kg$ é a massa do foguete no momento do lançamento, $\dot{m} = 13.3 \times 10^3\,kg/s$ é a taxa de consumo de combustível, $g = 9.81\,m/s^2$ a aceleração gravitacional e t o tempo medido a partir do lançamento.
Determine o instante de tempo t^* quando o foguete atinge a velocidade do som ($335\,m/s$).

Resolução Seguiremos a mesma ideia utilizada no Problema 7. Primeiramente, construímos uma função auxiliar para calcular parâmetros e, em seguida, definimos uma função `f(t)`.

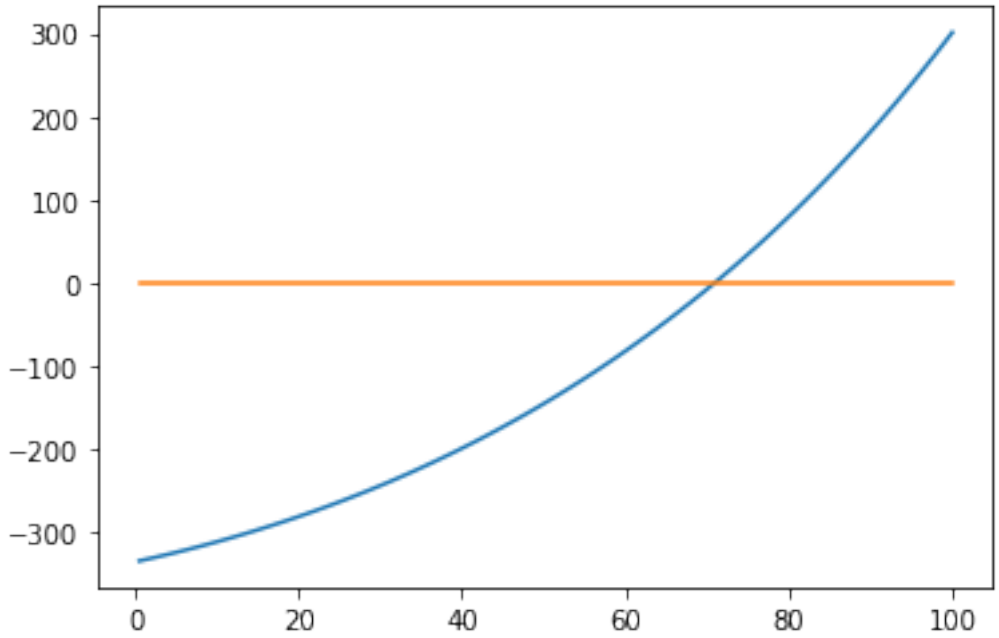
[29]:
```
# função para cálculo de parâmetros
def a(u,M0,m,g,v):
    return u,M0,m,g,v

# função do tempo
def f(t):
    escape,mass,fuel,grav,vel = a(u,M0,m,g,v)
    return escape*np.log(mass/(mass - fuel*t)) - g*t - vel
```

Definimos os parâmetros do problema.

[30]:
```
# parâmetros de entrada
u = 2510.0 # m/s
M0 = 2.8e6 # kg
m = 13.3e3 # kg/s
g = 9.81 # m/s2
v = 335.0 # m/s
```

Utilizaremos a análise gráfica para determinar o intervalo de refinamento da raiz.

[31]:
```
# análise gráfica
t = np.linspace(0.5,100,num=100)
plt.plot(t,f(t),t,f(t)*0);
```



Podemos verificar que a raiz está entre 60 e 80 segundos. Utilizaremos estes limitantes.

[32]:
```
# solução
tr = bisect(f,60,80)
print('Raiz: tr = %.2f s = %.2f min' % (tr,tr/60) )
```

```
Raiz: tr = 70.88 s = 1.18 min
```

O foguete rompe a barreira do som em 1 minuto e 18 segundos!

2 Code session 2

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import sympy as sy
```

2.1 Determinação de raízes

2.2 `newton`

A função `newton` localiza a raiz de uma função dentro de um intervalo dado usando o método de Newton. Os argumentos de entrada obrigatórios desta função são:

1. a função-alvo `f` (contínua)
2. a estimativa inicial `x0`

Parâmetros opcionais relevantes são:

- `fprime`: a derivada da função, quando disponível. Caso ela não seja especificada, o *método da secante* é usado.

- `fprime2`: a segunda derivada da função, quando disponível. Se ela for especificada, o *método de Halley* é usado.
- `tol`: tolerância (padrão: 1.48e-08)
- `maxiter`: número máximo de iterações (padrão: 50)
- `disp`: mostra erro se algoritmo não convergir (padrão: True)

O argumento de saída é:

- `x`: a estimativa para a raiz de `f`

Como importá-la?

```
from scipy.optimize import newton
```

[2]:
```
from scipy.optimize import newton
```

2.2.1 Problema 1

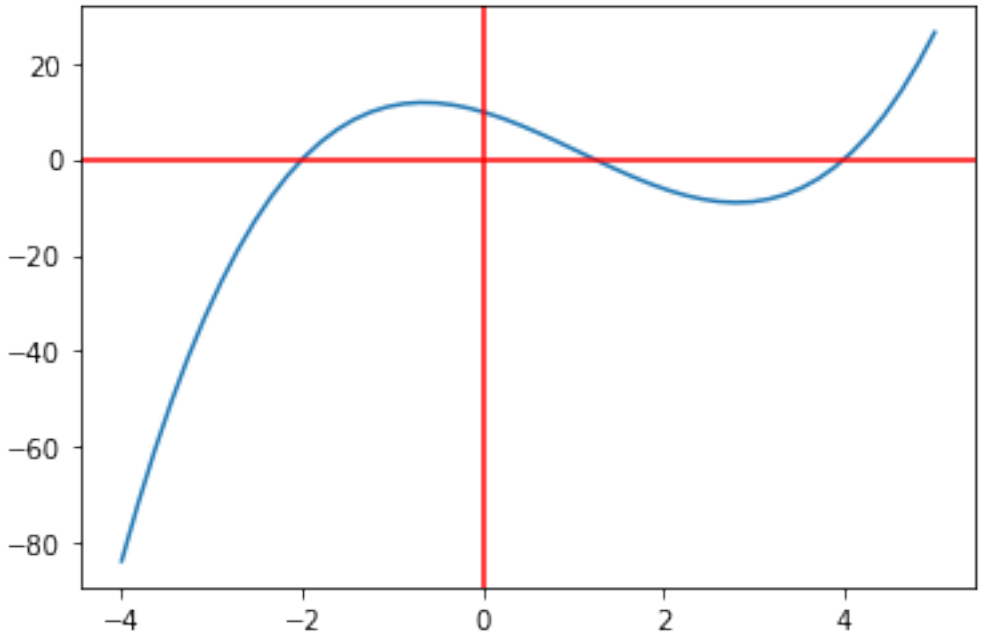
Encontre a menor raiz positiva (real) de $x^3 - 3.23x^2 - 5.54x + 9.84 = 0$ pelo método de Newton.

Resolução Definimos a função e sua primeira derivada.

[3]:
```
# função
def f(x):
    return x**3 - 3.23*x**2 - 5.54*x + 9.84

# 1a. derivada
def df(x):
    return 3*x**2 - 2*3.23*x - 5.54
```

Realizamos a análise gráfica.

[4]:
```
# analise gráfica
x = np.linspace(-4,5)
plt.plot(x,f(x));
plt.axhline(y=0,color='r');
plt.axvline(x=0,color='r');
```



Vamos realizar um estudo de diferentes estimativas iniciais e ver o que acontece.

Estimativa inicial: $x_0 = -1$

[5]:
```
# resolução com newton
x0 = -1
x = newton(f,x0,df) # raiz
print('Raiz: x = %f' % x)
```

```
Raiz: x = -2.000000
```

Estimativa inicial: $x_0 = 0$

[6]:
```
# resolução com newton
x0 = 0
x = newton(f,x0,df) # raiz
print('Raiz: x = %f' % x)
```

```
Raiz: x = 1.230000
```

Estimativa inicial: $x_0 = 3$

[7]:
```
# resolução com newton
x0 = 3
x = newton(f,x0,df) # raiz
print('Raiz: x = %f' % x)
```

```
Raiz: x = 4.000000
```

2.2.2 Problema 2

Determine a menor raiz não nula positiva de $\cosh(x)\cos(x) - 1 = 0$ dentro do intervalo $(4, 5)$.

Resolução: Primeiramente, vamos escrever a função `f(x)`.

[8]:
```
# função
f = lambda x: np.cosh(x)*np.cos(x) - 1
```

Para computar a primeira derivada, vamos utilizar computação simbólica. Veja no início deste notebook que inserimos a instrução

```
import sympy as sy
```

a qual nos permitirá utilizar objetos do módulo `sympy`.
Em primeiro lugar, devemos estabelecer uma variável simbólica `xs`.

[9]:
```
xs = sy.Symbol('x')
```

Em seguida, devemos utilizar as funções `cosh` e `cos` **simbólicas** para derivar `f`. Elas serão **chamadas de dentro do módulo sympy**.
Escrevemos a expressão simbólica para a derivada.

[10]:
```
d = sy.diff(sy.cosh(xs)*sy.cos(xs) - 1)
d
```

[10]: $-\sin(x)\cosh(x) + \cos(x)\sinh(x)$

Note que `d` é um objeto do módulo `sympy`

[11]:
```
type(d)
```

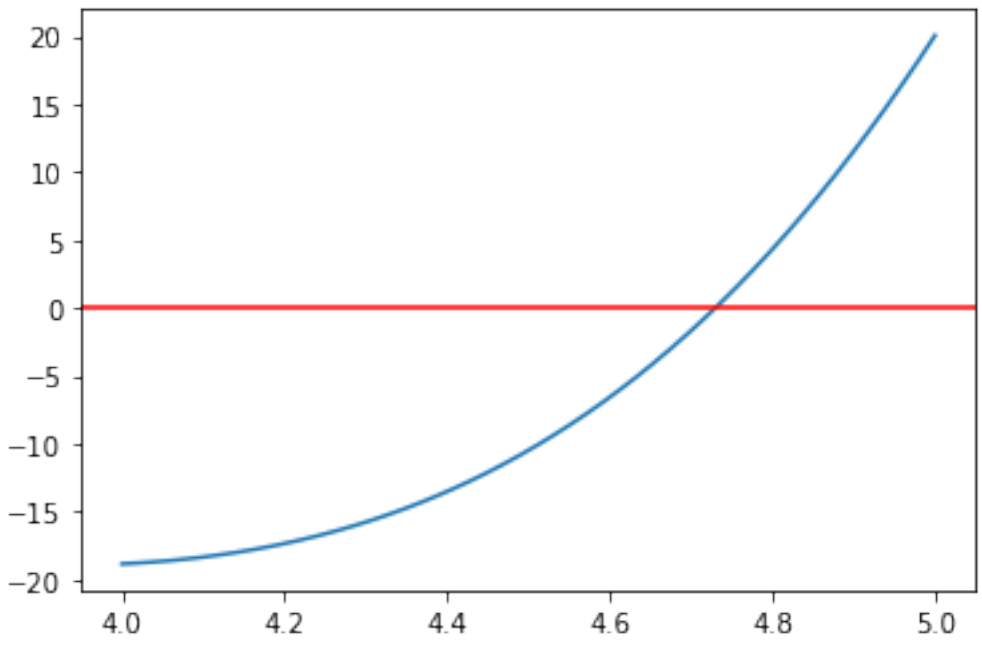
[11]: `sympy.core.add.Add`

Podemos agora aproveitar a expressão de `d` para criar nossa derivada.

[12]:
```
df = lambda x: - np.sin(x)*np.cosh(x) + np.cos(x)*np.sinh(x)
```

Agora, realizamos a análise gráfica.

[13]:
```
# analise gráfica
x = np.linspace(4,5)
plt.plot(x,f(x));
plt.axhline(y=0,color='r');
```



Agora, vamos resolver optando pela estimativa inicial $x_0 = 4.9$.

[14]:
```
# resolução com newton
x0 = 4.9
x = newton(f,x0,df) # raiz
print('Raiz: x = %f' % x)
```

```
Raiz: x = 4.730041
```

2.3 *Homework*

1. Reproduza os Problemas de 3 a 8 da *Code Session 1* resolvendo com o método `newton`.
2. Para os casos possíveis, determine a derivada. Caso contrário, utilize como método da Secante.
3. Pesquise sobre o método de Halley e aplique-o aos problemas usando também a função `newton`, mas avaliando-a também com a segunda derivada.

3 Code session 3

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

3.1 Determinação de raízes de polinômios

3.1.1 `roots`

A função `roots` computa as raízes de uma função dentro de um intervalo dado usando o método de Hörner. O único argumento de entrada desta função é

1. o *array* `p` com os coeficientes dos termos do polinômio.

$$P(x) = p_n x^n + p_{n-1} x^{n-1} + \ldots + p_1 x + p_0$$

O argumento de saída é:

- `x`: *array* com as raízes de $P(x)$.

Como importá-la?

```
from numpy import roots
```

Porém, como já fizemos uma importação do `numpy` acima, basta utilizarmos

```
np.roots(p)
```

3.1.2 Problema 1

Determine as raízes de $P(x) = 3x^3 + 7x^2 - 36x + 20$.

Resolução Para tornar claro, em primeiro lugar, vamos inserir os coeficientes de $P(x)$ em um *array* chamado `p`.

[2]:
```
p = np.array([3,7,-36,20])
```

Em seguida, fazemos:

[3]:
```
x = np.roots(p)
```

Podemos imprimir as raízes da seguinte forma:

[4]:
```
for i, v in enumerate(x):
    print(f'Raiz {i}: {v}')
```

```
Raiz 0: -5.0
Raiz 1: 1.9999999999999987
Raiz 2: 0.6666666666666669
```

3.1.3 `polyval`

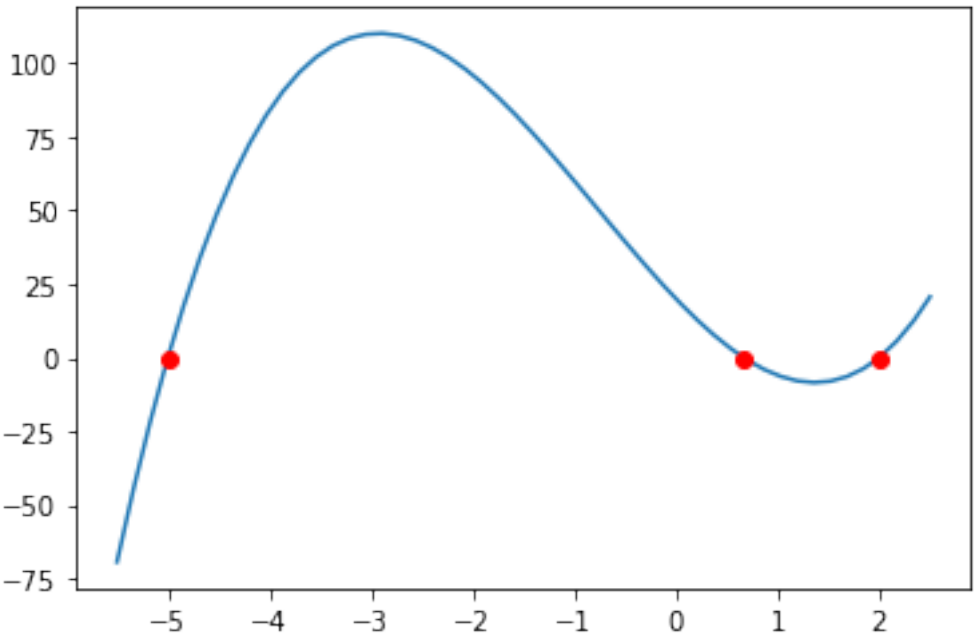
Podemos usar a função `polyval` do `numpy` para avaliar $P(x)$ em $x = x_0$. Verifiquemos, analiticamente, se as raízes anteriores satisfazem realmente o polinômio dado.

[5]:
```
for i in x:
    v = np.polyval(p,i)
    print(f'P(x) = {v}')
```

```
P(x) = 0.0
P(x) = -3.552713678800501e-14
P(x) = -7.105427357601002e-15
```

Note que as duas últimas raízes são "muito próximas" de zero, mas não exatamente zero. Podemos também fazer uma verificação geométrica plotando o polinômio e suas raízes.

[6]:
```
xx = np.linspace(np.min(x)-0.5,np.max(x)+0.5)
plt.plot(xx,np.polyval(p,xx));
for i in x:
    plt.plot(i,np.polyval(p,i),'or')
```



3.1.4 Problema 2

Determine as raízes de $P(x) = x^4 - 3x^2 + 3x$.

Resolução Resolvendo diretamente com `roots` e usando `polyval` para verificação, temos:

[7]:
```
# coeficientes e raízes
p = np.array([1,0,-3,3,0])
x = np.roots(p)
```

[8]:
```
# imprimindo as raizes
for i, v in enumerate(x):
    print(f'Raiz {i}: {v}')
```

```
Raiz 0: (-2.1038034027355357+0j)
Raiz 1: (1.051901701367768+0.5652358516771712j)
Raiz 2: (1.051901701367768-0.5652358516771712j)
Raiz 3: 0j
```

Note que, neste caso, as raízes são complexas.

3.1.5 Problema 3

Determine as raízes de $P(x) = x^5 - 30x^4 + 361x^3 - 2178x^2 + 6588x - 7992$.

Resolução

[9]:
```
# coeficientes e raízes
p = np.array([1,-30,361,-2178,6588,-7992])
x = np.roots(p)
```

[10]:
```
# imprimindo as raizes
for i, v in enumerate(x):
    print(f'Raiz {i}: {v}')
```

```
Raiz 0: (6.000000000009944+0.9999999999996999j)
Raiz 1: (6.000000000009944-0.9999999999996999j)
Raiz 2: (6.00026575921113+0j)
Raiz 3: (5.999867120384507+0.00023015565268626 68j)
Raiz 4: (5.999867120384507-0.00023015565268626 68j)
```

4 Code session 4

4.1 `fsolve`

A função `fsolve` do submódulo `scipy.optimize` pode ser usada como método geral para busca de raízes de equações não-lineares escalares ou vetoriais.
Para usar `fsolve` em uma equação escalar, precisamos de, no mínimo:

- uma função que possui pelo menos um argumento

- estimativa inicial para a raiz

Para equações vetoriais (sistemas), precisamos de mais argumentos. Vejamos o exemplo do paraquedista:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

t = 12.0
v = 42.0
m = 70.0
g = 9.81

def param(t,v,m,g):
    return [t,v,m,g]

def fun(c):
    p = param(t,v,m,g)
    return p[3]*p[2]/c*(1 - np.exp(-c/p[2]*p[0])) - p[1]

# estimativa inicial
c0 = -1000.0

# raiz
c_raiz = fsolve(fun,c0)

# impressao (estilo Python 2)
print('Minha raiz é %.6f' % c_raiz)

# impressao (estilo Python 3)
print("Minha raiz é {0:.6f}".format(c_raiz[0]))
```

```
Minha raiz é 15.127432
Minha raiz é 15.127432
```

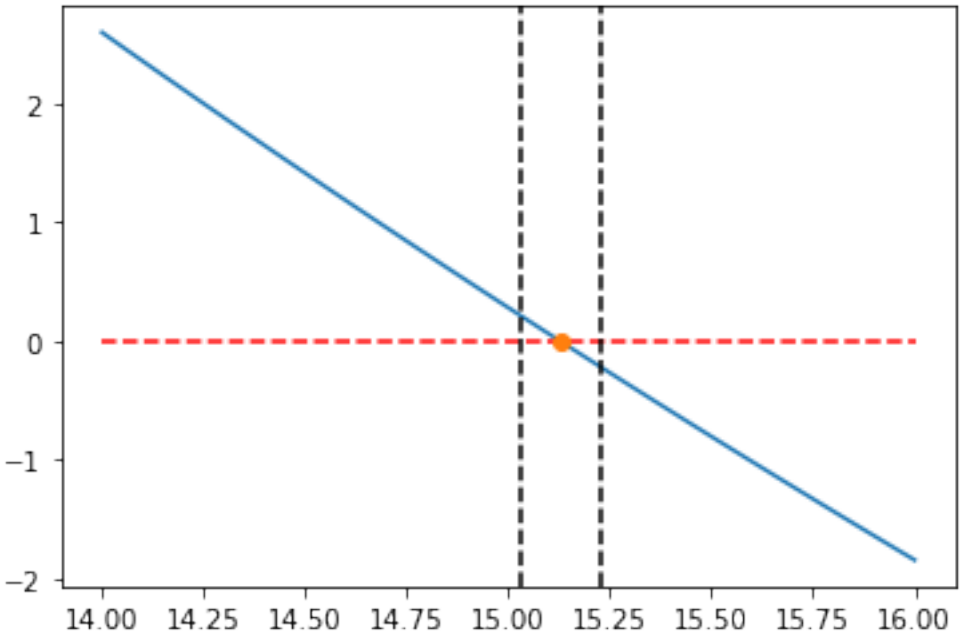
4.1.1 Como incorporar tudo em uma só função

[2]:
```
def minha_fun(t,v,m,g,c0):

    p = [t,v,m,g]
    f = lambda c: p[3]*p[2]/c*(1 - np.exp(-c/p[2]*p[0])) - p[1]
    c_raiz = fsolve(f,c0)
    print("---> Minha raiz é {0:.6f}".format(c_raiz[0]))
    return f,c_raiz[0]
```

[3]:
```
fc,c_raiz = minha_fun(t,v,m,g,c0)
```

```
---> Minha raiz é 15.127432
```

[4]:
```
a,b = 14,16
c = np.linspace(a,b,100)

delta = 0.1
plt.plot(c,fc(c),c,0*c,'r--',c_raiz,fc(c_raiz),'o');
plt.axvline(c_raiz - delta,c='k',ls='--');
plt.axvline(c_raiz + delta,c='k',ls='--');
```



5 Code session 5

[1]:
```
import numpy as np
```

5.1 `linalg.solve`

A função `solve` é o método mais simples disponibilizado pelos módulos `numpy` e `sympy` para resolver sistemas matriciais lineares. Como a função pertence ao escopo da Álgebra Linear, ela está localizada em submódulo chamado `linalg`. `solve` calculará a solução exata do sistema como um método direto se a matriz do sistema for determinada (quadrada e sem colunas linearmente dependentes). Se a matriz for singular, o método retorna um erro. Se for de posto deficiente, o método resolve o sistema linear usando um algoritmo de mínimos quadrados.
Os argumentos de entrada obrigatórios desta função são:

1. a matriz `A` dos coeficientes
2. o vetor independente `b`

O argumento de saída é:

- `x`: o vetor-solução do sistema.