Equações não Lineares

31 de janeiro de 2022

1 Resolvendo Equações não Lineares

Vamos tratar aqui sobre como encontrar soluções para a equação não linear

$$f(x) = 0\ , \tag{1}$$

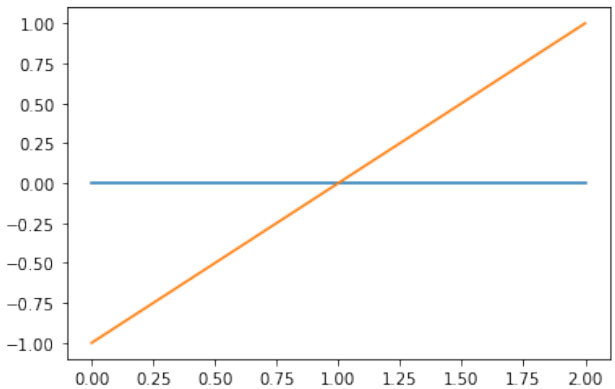
com a variável x estando no intervalo $[a, b]$. Vamos assumir que a função $f(x)$ é contínua nesse intervalo. Uma solução da equação acima será chamada de *raiz* da equação $f(x) = 0$ ou *zero* da função $f(x)$, e será denotada por x^*.

Além de equações não lineares aparecerem frequentemente na modelagem de problemas reais, esse estudo inicial irá nos possibilitar entender vários conceitos e detalhes que aparecem em problemas números mais complexos, com os quais iremos nos deparar mais adiante em nosso curso.

Alguns exemplos de funções:

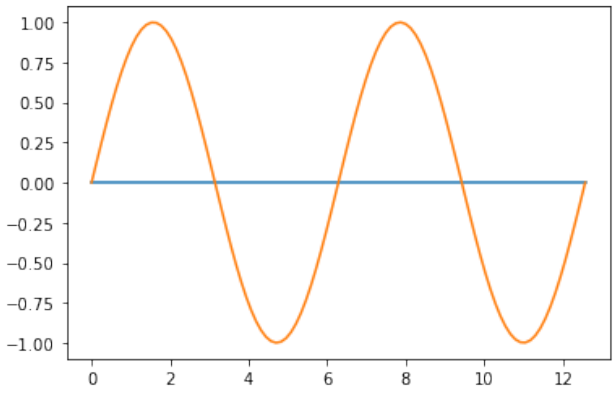
1) $f(x) = x - 1$, para $0 \leq x \leq 2$. Nesse caso temos uma raiz, $x^* = 1$.

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

x = np.linspace(0.0,2.0,100)
zero = 0.0*x
f = x - 1.0
plt.plot(x,zero)
plt.plot(x,f)
plt.show()
```



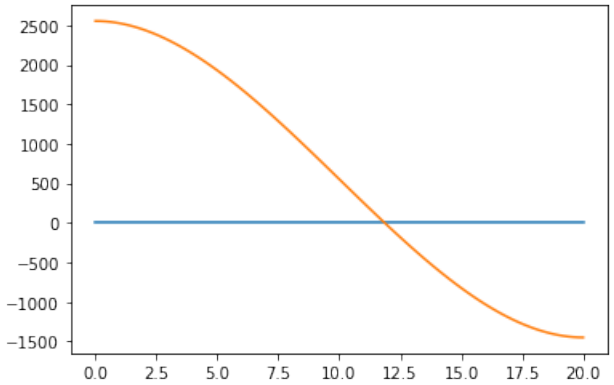
2) $f(x) = \sin(x)$. No intervalo $[\pi/2, 3\pi/2]$ temos apenas uma raiz, $x^* = \pi$. No intervalo $[0, 4\pi]$ temos 5 raízes.

[2]:
```
x = np.linspace(0.0,4.0*np.pi,100)
f = np.sin(x)
plt.plot(x,zero)
plt.plot(x,f)
plt.show()
```



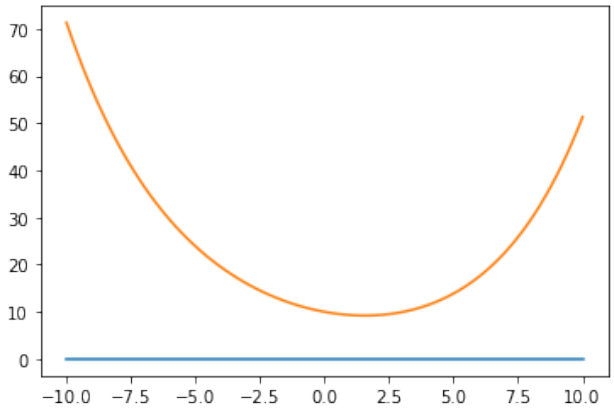
3) $f(x) = x^3 - 30x^2 + 2552$, em $[0, 20]$. Pelo gráfico abaixo, a suspeita é de que temos uma raiz nesse intervalo.

[3]:
```
x = np.linspace(0.0,20,100)
f = x**3.0 - 30.0*x**2.0 + 2552
plt.plot(x,zero)
plt.plot(x,f)
plt.show()
```



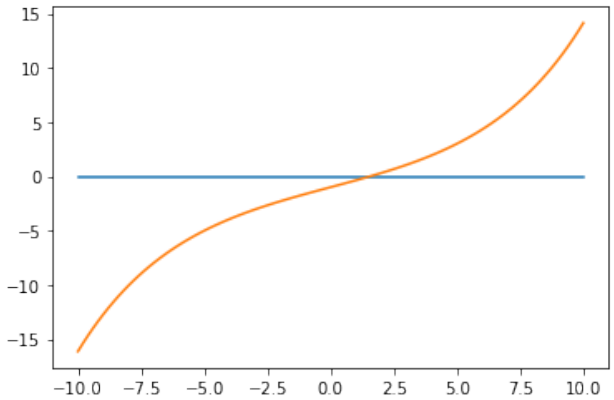
4) $f(x) = 10\cosh(x/4) - x$, para $-10 \leq x \leq 10$. A equação $f(x) = 0$ para esse caso não tem uma raiz real. Porém, percebe-se que essa função tem um mínimo. Assim, espera-se que $f'(x) = 0$ tenha raiz.

[4]:
```
x = np.linspace(-10,10,100)
f = 10.0*np.cosh(x/4.0) - x
plt.plot(x,zero)
plt.plot(x,f)
plt.show()
```



5) $f(x) = 2.5\sinh(x/4) - 1$, para $-10 \leq x \leq 10$. Pelo gráfico abaixo, essa função tem pelo menos um zero.

[13]:
```
x = np.linspace(-10,10,100)
f = 2.5*np.sinh(x/4.0) - 1.0
plt.plot(x,zero)
plt.plot(x,f)
plt.show()
```



Vimos, portanto, exemplos de equações que com uma raiz, várias raízes ou com nenhuma raiz. É importante ter algum tipo de noção sobre o comportamento da equação que se pretende resolver numericamente.

1.1 Métodos Iterativos

Apenas alguns tipos especiais de equações não lineares possuem soluções dadas por fórmulas simples, fechadas (por exemplo, a equação do segundo grau). Na maioria dos casos, é necessário recorrer a ***métodos iterativos*** para encontrar as raízes de uma dada equação não linear.

Um **método iterativo** é aquele que começa com uma **iteração (*chute*) inicial** x_0 e, a partir de um dado procedimento, vai gerando uma sequência de aproximações x_1, x_2, x_3, $\cdots$, x_k, $\cdots$. Se tudo estiver funcionando como o planejado, esses valores se aproximam (*convergem*) cada vez mais da raiz da equação. Para dar certo, devemos ter algum conhecimento sobre a localização das raízes. Podemos chegar em uma raiz ou em outra ou em nenhuma, dependendo do *chute inicial*.

Algumas ideias: 1. podemos simplesmente plotar o gráfico da função $f(x)$.

1. podemos examinar o valor de $f(x)$ em diferentes intervalos $[a, b]$. Se $f(a) \cdot f(b) < 0$ significa que a função $f(x)$ muda de sinal no intervalo $[a, b]$. Como a função é contínua, existe, pelo Teorema do Valor Intermediário, algum $c = x^*$ nesse intervalo para o qual $f(c) = 0$.

É possível encontrar aproximações para as raízes de uma equação com esses métodos, mas esses métodos não serão muito úteis quando estivermos tratando de várias equações com várias incógnitas, por exemplo.

Veremos alguns métodos mais sofisticados e mais generalizáveis aqui.

1.2 Como parar um Método Iterativo

Um método iterativo, tipicamente, começa com uma solução inicial x_0 e gera uma sequência $x_1, x_2, \cdots, x_k, \cdots$. Não é esperado que esse procedimento leve à solução **exata**, mas sim a uma solução aproximada. A sequência de iterações converge se o valor de $|f(x_k)|$ e/ou o valor de $|x_k - x_{k-1}|$ decresce para 0 na medida em que k cresce.

Assim, podemos usar um ou mais dos seguintes critérios de parada (ou seja, para finalizar o processo iterativo), após n iterações: 1. $|x_n - x_{n-1}| < atol$ 2. $|x_n - x_{n-1}| < rtol\ |x_n|$ 3. $|f(x_n)| < ftol$

Os valores $atol$, $rtol$ e $ftol$ são **tolerâncias** especificadas por quem está resolvendo o problema. Geralmente, o segundo critério, que leva em consideração o erro relativo, é o melhor.

Na hora de programar, é interessante colocar um valor máximo para o número de iterações que o seu código vai realizar, para que ele não entre em *loop* infinito caso não haja convergência.

1.3 Propriedades Desejadas nos Métodos

Espera-se que o algoritmo utilizado para encontrar raízes de uma equação seja:

- **Eficiente**. O algoritmo deve ser rápido. O ideal é que ele convirja para a resposta correta em poucas iterações. Além disso, é interessante que o algoritmo utilize o menor número possível de valores de $f(x)$ e de $f'(x)$, pois essas funções podem ser de difícil acesso (caras).

- **Robusto**. O algoritmo deve falhar apenas em raríssimas situações. E quando isso ocorrer, o algoritmo deve reconhecer que falhou e avisar o usuário. Ou seja, o algoritmo deve ser confiável.
- **Generalizável**. O algoritmo deve possibilitar a solução de diferentes tipos de equações e também deve ser aplicável para casos com várias equações e várias incógnitas.

Vamos estudar 3 algoritmos aqui: **Método da Bissecção, Método do Ponto Fixo e Método de Newton.** Cada um possui vantagens e desvatagens: a escolha depende do problema. Com tudo funcionando bem, o Método de Newton é o mais eficiente.

2 Método da Bissecção

É um método simples e seguro que requer poucas hipóteses com relação à função $f(x)$. No entanto, é um método lento e de difícil generalização.

A ideia do método é a seguinte: para uma dada $f(x)$ contínua, nós sabemos de um intervalo $[a, b]$ em que f muda de sinal, ou seja, $f(a) \cdot f(b) < 0$. Pelo Teorema do Valor Intermediário, existe uma raiz x^* tal que $a \leq x^* \leq b$. Assim, calcule $f(p)$, com $p = (a + b)/(2)$. Se $f(a) \cdot f(p) < 0$, então a raiz está no intervalo $[a, p]$, e podemos fazer $b \leftarrow p$ para o próximo passo. Se $f(a) \cdot f(p) > 0$, então a raiz deve estar no intervalo $[p, b]$, e aí podemos fazer $a \leftarrow p$.

O procedimento descrito no parágrafo acima corresponde a uma iteração, na qual o intervalo em que x^* deve estar foi reduzido pela metade. Continuando as iterações, no passo k, a resposta aproximada dada pelo método será o ponto médio do intervalo reduzido. Assim, com $x_0 = (a + b)/2$, temos

$$|x^* - x_0| < \frac{b-a}{2} .$$

Após 1 iteração temos

$$|x^* - x_1| < \frac{b-a}{2} \cdot \frac{1}{2} .$$

De maneira geral, após k iterações teremos

$$|x^* - x_k| < \frac{b-a}{2} \cdot 2^{-k} .$$

Para parar o processo iterativo, devemos ter $|x^* - x_k| < atol$. Podemos calcular o número de iterações necessárias assumindo que o erro máximo seja menor que a tolerânica. Assim,

$$\frac{b-a}{2} \cdot 2^{-k} < atol .$$

Ou seja:

$$k > \log_2 \left(\frac{b-a}{2atol} \right) .$$

Algoritmo: Método da Bissecção

Entradas: $f, a, b, atol$. É necessário que $f(a) \cdot f(b) < 0$.

Saída: valor x tal que $|x^* - x| < atol$.

1. $kmax \leftarrow \log_2\left(\frac{b-a}{2atol}\right)$
2. $k \leftarrow 0$
3. enquanto $k <= kmax + 1$, faça:
    1. $p_k \leftarrow (a+b)/2$
    2. se $f(a) \cdot f(p_k) < 0$ então $b \leftarrow p_k$
    3. se $f(a) \cdot f(p_k) > 0$ então $a \leftarrow p_k$
    4. $k \leftarrow k + 1$
4. $x \leftarrow (a+b)/2$

Fim do Algoritmo

Comentário. Um **Algoritmo** é um procedimento que descreve uma sequência finita de passos a serem feitos, em uma ordem específica. O objetivo do algoritmo é resolver um problema. Utilizaremos aqui **Pseudocódigos** para descrever os algoritmos. Os algoritmos aqui apresentados podem ser implementados em qualquer linguagem de programação.

Abaixo temos um código em Python que implementa o algoritmo acima para algumas funções f:

$$f(x) = x^3 - 30x^2 + 2552$$

$$f(x) = 2.5\sinh(x/4) - 1$$

$$f(x) = e^{-x} - x$$

[30]:
```
def f(y):    # Alguns exemplos de funções f
    #return y**3.0 - 30.0*(y**2.0) + 2552.0    # Usar a = 0 e b = 20
    return 2.5*np.sinh(y/4.0) - 1.0    # Usar a = -10 e b = 10
    #return np.exp(-y) - y    # Usar a = 0.0 e b = 2.0

a, b, atol = -10.0, 10.0, 1.e-8
kmax = np.log2((b-a)/(2.0*atol))
k = 0
while k <= kmax+1:
    p = (a+b)/2.0
    if f(a)*f(p) < 0.0:
        b = p
    else:
```

```
        a = p
    k = k + 1
    print(f'Iteração = {k:3d}, Valor de p = {p:10.9f}, Valor de f(p) = {f(p):14.
 ↪13}')
x = (a+b)/2.0
```

```
Iteração =   1, Valor de p = 0.000000000, Valor de f(p) =           -1.0
Iteração =   2, Valor de p = 5.000000000, Valor de f(p) = 3.004797700752
Iteração =   3, Valor de p = 2.500000000, Valor de f(p) = 0.6662306611415
Iteração =   4, Valor de p = 1.250000000, Valor de f(p) = -0.2059721097161
Iteração =   5, Valor de p = 1.875000000, Valor de f(p) = 0.2152643004326
Iteração =   6, Valor de p = 1.562500000, Valor de f(p) = 0.001587936562512
Iteração =   7, Valor de p = 1.406250000, Valor de f(p) = -0.1028766244312
Iteração =   8, Valor de p = 1.484375000, Valor de f(p) = -0.05082539037892
Iteração =   9, Valor de p = 1.523437500, Valor de f(p) = -0.02466523486359
Iteração =  10, Valor de p = 1.542968750, Valor de f(p) = -0.01155043241102
Iteração =  11, Valor de p = 1.552734375, Valor de f(p) = -0.004984213303852
Iteração =  12, Valor de p = 1.557617188, Valor de f(p) = -0.001698882163056
Iteração =  13, Valor de p = 1.560058594, Valor de f(p) = -5.565905442562e-05
Iteração =  14, Valor de p = 1.561279297, Valor de f(p) = 0.0007660921522401
Iteração =  15, Valor de p = 1.560668945, Valor de f(p) = 0.00035520490324
Iteração =  16, Valor de p = 1.560363770, Valor de f(p) = 0.0001497700135882
Iteração =  17, Valor de p = 1.560211182, Valor de f(p) = 4.70547519511e-05
Iteração =  18, Valor de p = 1.560134888, Valor de f(p) = -4.30233313542e-06
Iteração =  19, Valor de p = 1.560173035, Valor de f(p) = 2.137616393227e-05
Iteração =  20, Valor de p = 1.560153961, Valor de f(p) = 8.536904029466e-06
Iteração =  21, Valor de p = 1.560144424, Valor de f(p) = 2.117282604797e-06
Iteração =  22, Valor de p = 1.560139656, Valor de f(p) = -1.092525975688e-06
Iteração =  23, Valor de p = 1.560142040, Valor de f(p) = 5.123781368077e-07
Iteração =  24, Valor de p = 1.560140848, Valor de f(p) = -2.900739638489e-07
Iteração =  25, Valor de p = 1.560141444, Valor de f(p) = 1.111520755437e-07
Iteração =  26, Valor de p = 1.560141146, Valor de f(p) = -8.946094687268e-08
Iteração =  27, Valor de p = 1.560141295, Valor de f(p) = 1.084556355835e-08
Iteração =  28, Valor de p = 1.560141221, Valor de f(p) = -3.930769176819e-08
Iteração =  29, Valor de p = 1.560141258, Valor de f(p) = -1.423106410492e-08
Iteração =  30, Valor de p = 1.560141277, Valor de f(p) = -1.692750384308e-09
Iteração =  31, Valor de p = 1.560141286, Valor de f(p) = 4.57640658702e-09
```

No código acima, poderíamos ter usado o próprio valor de f como critério de parada. Poderíamos, também, ter deixado o código mais eficiente, parando o *loop* caso a resposta estivesse já dentro da tolerância aceitável. Seria interessante, ainda, colocar todo o código dentro de uma função, com *inputs* a, b e $atol$, e *output* x. Vou deixar essas possíveis modificações para vocês.

Importante: não dê ctrl c ctrl v nos códigos aqui das aulas nem de qualquer outro lugar. Escreva você mesmo o código, linha por linha, para realmente entender como o código funciona. Faça vários testes e modificações para ver os resultados.

A vantagem do Método da Bissecção é que a raiz está presa (delimitada), então o método sempre converge. A maior desvantagem do método é que ele possui convergência lenta: são necessárias

muitas iterações.

3 Método do Ponto Fixo

Nosso problema aqui é resolver a equação

$$f(x) = 0 \ .$$

Podemos reescrever essa equação como sendo

$$x = g(x) \ ,$$

com $g(x)$ contínua.

Essa função $g(x)$ é tal que x satisfaz $f(x) = 0$ se, e somente se, $x = g(x)$. Dada essa última forma, estamos procurando um **ponto fixo**, ou seja, um ponto x^* tal que

$$x^* = g(x^*) \ .$$

No **Método do Ponto Fixo**, nós começamos com um chute inicial x_0 e calculamos os valores x_1, x_2, $\cdots$, x_k, $\cdots$ por meio do processo iterativo dado por

$$x_{k+1} = g(x_k), \text{ para } k = 0, 1, 2, \cdots$$

Se essa sequência converge, o limite desse processo iterativo é um ponto fixo.

Para uma mesma função $f(x)$ podemos escolher diferentes funções $g(x)$, que vão dar origem ao processo iterativo. Essa escolha deve ser feita com cuidado.

Algoritmo: Método do Ponto Fixo

Entradas: função $g(x)$; aproximação inicial x_0; tolerância $atol$.

Saída: valor aproximado da solução

1. Comece com uma aproximação inicial x_0
2. $k \leftarrow 0$
3. $dif \leftarrow 2 \cdot atol$
4. enquanto $dif >= atol$, faça:
    1. $x_{k+1} \leftarrow g(x_k)$
    2. $dif \leftarrow |x_{k+1} - x_k|$
    3. $k \leftarrow k + 1$

Fim do Algoritmo

Exemplo. Abaixo temos um exemplo em Python que implementa o algoritmo acima para resolver a equação

$$f(x) = x - e^{-x} = 0$$

com

$$g(x) = e^{-x}\,.$$

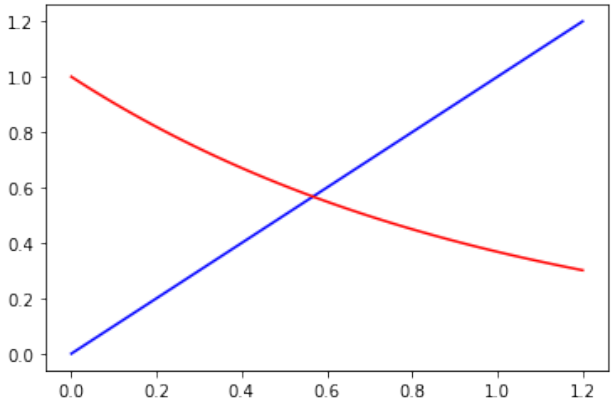
[19]:
```
def g(y):
    return np.exp(-y)

x = 1.0    # Esse é o chute inicial
atol = 1.e-8
k = 0
dif = 2.0*atol    # O valor inicial de dif deve ser maior que atol. Por quê?
while dif >= atol:
    x_new = g(x)
    dif = np.abs(x_new - x)    # Critério de saída
    x = x_new
    k += 1
    print('Na iteração k = %2.2i, o valor de x é %12.10f'%(k,x))
```

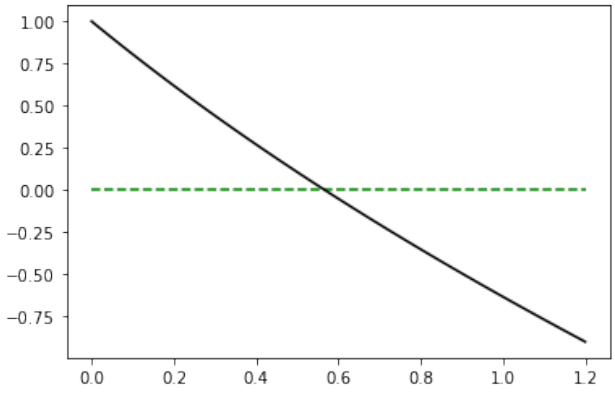
```
Na iteração k = 01, o valor de x é 0.3678794412
Na iteração k = 02, o valor de x é 0.6922006276
Na iteração k = 03, o valor de x é 0.5004735006
Na iteração k = 04, o valor de x é 0.6062435351
Na iteração k = 05, o valor de x é 0.5453957860
Na iteração k = 06, o valor de x é 0.5796123355
Na iteração k = 07, o valor de x é 0.5601154614
Na iteração k = 08, o valor de x é 0.5711431151
Na iteração k = 09, o valor de x é 0.5648793474
Na iteração k = 10, o valor de x é 0.5684287250
Na iteração k = 11, o valor de x é 0.5664147331
Na iteração k = 12, o valor de x é 0.5675566373
Na iteração k = 13, o valor de x é 0.5669089119
Na iteração k = 14, o valor de x é 0.5672762322
Na iteração k = 15, o valor de x é 0.5670678984
Na iteração k = 16, o valor de x é 0.5671860501
Na iteração k = 17, o valor de x é 0.5671190401
Na iteração k = 18, o valor de x é 0.5671570440
Na iteração k = 19, o valor de x é 0.5671354902
Na iteração k = 20, o valor de x é 0.5671477143
Na iteração k = 21, o valor de x é 0.5671407815
Na iteração k = 22, o valor de x é 0.5671447133
Na iteração k = 23, o valor de x é 0.5671424834
Na iteração k = 24, o valor de x é 0.5671437481
```

```
Na iteração k = 25, o valor de x é 0.5671430308
Na iteração k = 26, o valor de x é 0.5671434376
Na iteração k = 27, o valor de x é 0.5671432069
Na iteração k = 28, o valor de x é 0.5671433378
Na iteração k = 29, o valor de x é 0.5671432636
Na iteração k = 30, o valor de x é 0.5671433056
Na iteração k = 31, o valor de x é 0.5671432818
Na iteração k = 32, o valor de x é 0.5671432953
Na iteração k = 33, o valor de x é 0.5671432876
```

Podemos dar uma olhada nos gráficos de f e de g para entender um pouco melhor o que está acontecendo.

[8]:
```
x = np.linspace(0.0,1.2,100)
funcao_g = g(x)
funcao_f = g(x) - x
zero = 0.0*x

plt.plot(x,x,'b-')
plt.plot(x,funcao_g,'r-')
plt.show()
plt.plot(x,zero,'g--')
plt.plot(x,funcao_f,'k-')
plt.show()
```





Algumas perguntas:

1. existe um ponto fixo x^* no intervalo $[a, b]$?
2. esse ponto é único?
3. a sequência gerada pelo método converge para a raiz x^*?

Essas respostas são dadas pelo teorema abaixo.

Teorema do Ponto Fixo: se $g(x)$ é contínua no intervalo $[a, b]$ e $a \leq g(x) \leq b$ para todo $x \in [a, b]$, então existe um ponto fixo x^* no intervalo $[a, b]$. Se, além disso, a derivada g' existe e existe também uma constante positiva $\rho < 1$ tal que a derivada satisfaça

$$|g'(x)| \leq \rho \qquad \forall\, x \in (a, b)\ ,$$

então o ponto fixo x^* é único no intervalo e o método converge para essa solução.

A convergência do Método do Ponto Fixo pode ser analisada da seguinte maneira.

Considere o processo iterativo

$$x_{k+1} = g(x_k)\ .$$

Seja x^* a solução e seja $e_k = x_k - x^*$ o erro na iteração k.

Subtraindo $x^* = g(x^*)$ da equação acima resulta

$$x_{k+1} - x^* = e_{k+1} = g(x_k) - g(x^*)\ .$$

Podemos representar $g(x_k)$ por meio de um série de Taylor em torno de x^* como

$$g(x_k) = g(x^*) + g'(x^*)(x_k - x^*) + \cdots$$

Truncando a série depois do termo de primeira ordem temos

$$x_{k+1} - x^* = g(x_k) - g(x^*) \approx g'(x^*)(x_k - x^*) \ .$$

Ou seja:

$$e_{k+1} \approx g'(x_k)e_k$$

$$\left|\frac{e_{k+1}}{e_k}\right| \approx |g'(x_k)|$$

Essa equação é importante em qualquer método iterativo, pois ela nos diz se o erro está diminuindo ou não. Para que o método seja convergente, o erro tem que diminuir na medida em que o número de iterações aumenta.

Assim, nesse caso específico, o Método do Ponto Fixo é convergente se

$$|g'(x_k)| \leq \rho < 1 \ .$$

Temos

$$|x_1 - x^*| \leq \rho|x_0 - x^*|$$

$$|x_2 - x^*| \leq \rho|x_1 - x^*| \leq \rho^2|x_0 - x^*|$$

$$|x_3 - x^*| \leq \rho|x_2 - x^*| \leq \rho^2|x_1 - x^*| \leq \rho^3|x_0 - x^*|$$

Como $\rho < 0$, $\rho^k \to 0$ na medida em que $k \to \infty$. **O erro diminui em cada iteração**. De maneira geral temos

$$|x_k - x^*| \leq \rho|x_{k-1} - x^*| \leq \cdots \leq \rho^k|x_0 - x^*|$$

Quanto menor o valor de ρ mais rápida é a convergência do método. ρ é chamado de **Taxa de Convergência** do método.

Exemplo. Vamos encontrar os zeros da função

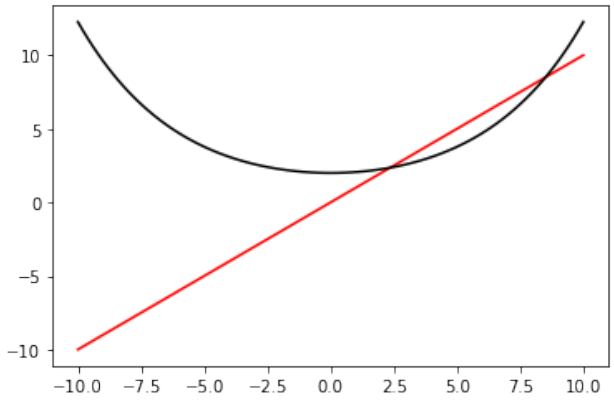
$$f(x) = 2\cosh(x/4) - x \ .$$

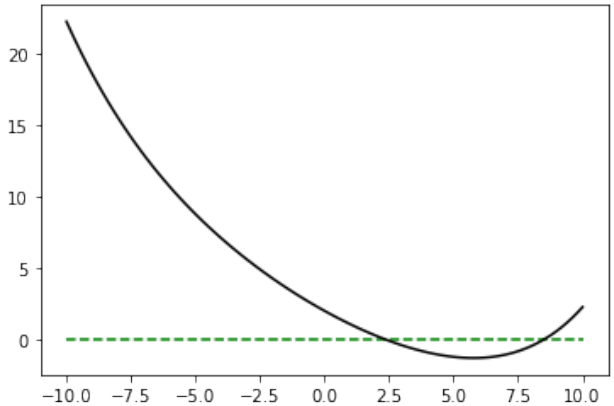
Vamos primeiro definir

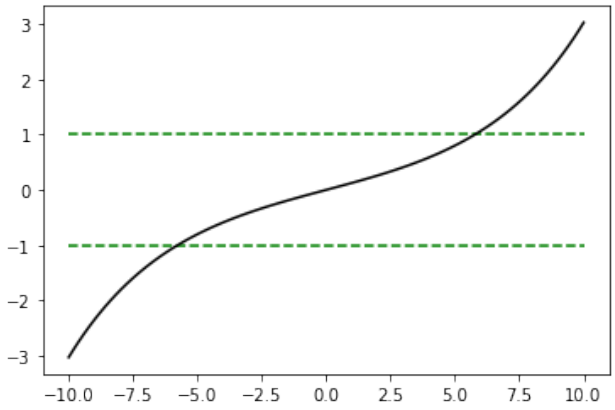
$$g(x) = 2\cosh(x/4)$$

e vamos plotar essas funções.

[9]:
```
x = np.linspace(-10.0,10.0,100)
funcao_g = 2.0*np.cosh(x/4.0)
funcao_g_linha = 0.5*np.sinh(x/4.0)
funcao_f = 2.0*np.cosh(x/4.0) - x
zero = 0.0*x

plt.plot(x,x,'r-')
plt.plot(x,funcao_g,'k-')
plt.show()

plt.plot(x,zero,'g--')
plt.plot(x,funcao_f,'k-')
plt.show()

plt.plot(x,np.ones(100),'g--')
plt.plot(x,-np.ones(100),'g--')
plt.plot(x,funcao_g_linha,'k-')
plt.show()
```







Pelo gráfico é possível ver que temos duas raízes nesse intervalo.

Implementando novamente o algoritmo do Método do Ponto Fixo temos:

[31]:
```
def g(y):
    return 2.0*np.cosh(y/4.0)

# Testar 3.0, 5.0, 7.0, 8.0, 9.0, -7.5, -9.0
x = 1.0
atol = 1.e-8
k = 0
dif = 2.0*atol
while dif >= atol:
    x_new = g(x)
    dif = np.abs(x_new - x)
    x = x_new
    k += 1
    print('Na iteração k = %2.2i, o valor de x é %12.10f'%(k,x))
```

```
Na iteração k = 01, o valor de x é 2.0628261998
Na iteração k = 02, o valor de x é 2.2719000085
Na iteração k = 03, o valor de x é 2.3313617243
Na iteração k = 04, o valor de x é 2.3494290319
Na iteração k = 05, o valor de x é 2.3550213996
Na iteração k = 06, o valor de x é 2.3567621356
Na iteração k = 07, o valor de x é 2.3573049143
Na iteração k = 08, o valor de x é 2.3574742494
Na iteração k = 09, o valor de x é 2.3575270871
Na iteração k = 10, o valor de x é 2.3575435750
Na iteração k = 11, o valor de x é 2.3575487201
Na iteração k = 12, o valor de x é 2.3575503256
Na iteração k = 13, o valor de x é 2.3575508266
Na iteração k = 14, o valor de x é 2.3575509830
Na iteração k = 15, o valor de x é 2.3575510317
```

```
Na iteração k = 16, o valor de x é 2.3575510470
Na iteração k = 17, o valor de x é 2.3575510517
```

Note que dependendo do chute inicial o método pode convergir para a raiz menor ou divergir, mas o método não converge para a raiz maior.

Note que a raiz maior existe, mesmo que o método do ponto fixo utilizado não 'veja' essa solução. Para encontrá-la temos que usar outro método ou então tentar definir uma função $g(x)$ diferente.

4 Método de Newton

Dentre os métodos que estamos estudando, esse é o método mais rápido e eficiente. O princípio utilizado em sua dedução pode ser generalizado para problemas mais complexos.

Queremos achar os zeros de uma função não linear $f(x)$. Vamos assumir que f é contínua, assim como a primeira e a segunda derivada de f. Vamos assumir também que $f'(x) = df/dx$ pode ser facilmente calculada (depois veremos o que fazer quando isso não é verdade).

Usando série de Taylor, podemos fazer uma expansão de $f(x)$ em torno de x_k, onde x_k é o resultado da iteração de número k. Assim,

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + f''(\xi)(x - x_k)^2/2 \ ,$$

em que ξ é um valor (desconhecido) entre x e x_k.

Agora, na equação acima, faça $x = x^*$, para o qual $f(x^*) = 0$ (x^* é uma raiz). Se a função f fosse linear, então f'' seria igual a zero e teríamos simplesmente

$$0 = f(x_k) + f'(x_k)(x^* - x_k) \ ,$$

resultando em

$$x^* = x_k - \frac{f(x_k)}{f'(x_k)} \ .$$

A ideia do Método de Newton é exatamente usar essa equação como fórmula para a iteração:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \ , \quad k = 0, 1, 2, \cdots$$

Isso corresponde a simplesmente ignorar o termo de segunda ordem ao definir as iterações. Como x_k vai se aproximando de x^*, nós esperamos que $(x^* - x_k)^2$ seja pequeno quando comparado aos outros termos da equação.

Algoritmo: Método de Newton

Entradas: função f e sua derivada f'; aproximação inicial x_0; tolerância $atol$.

Saída: valor aproximado da solução

1. Comece com uma aproximação inicial x_0
2. $k \leftarrow 0$
3. $dif \leftarrow 2 \cdot atol$
4. enquanto $dif >= atol$, faça:
    1. $x_{k+1} \leftarrow x_k - \frac{f(x_k)}{f'(x_k)}$
    2. $dif \leftarrow |x_{k+1} - x_k|$
    3. $k \leftarrow k + 1$

Fim do Algoritmo

Exemplo: vamos encontrar os zeros da função $f(x) = 2\cosh(x/4) - x$ utilizando uma implementação do algoritmo acima.

[42]:
```
def f(y):
    return 2.0*np.cosh(y/4.0) - y

def f_linha(y):
    return 0.5*np.sinh(x/4.0) - 1.0

x = 1.0    # 2.0, 10.0
atol = 1.e-14
k = 0
dif = 2.0*atol
while dif >= atol:
    x_new = x - f(x)/f_linha(x)
    dif = np.abs(x_new - x)
    x = x_new
    k += 1
    print('Na iteração k = %2.2i, o valor de x é %20.18f'%(k,x))
```

```
Na iteração k = 01, o valor de x é 2.216474409179484795
Na iteração k = 02, o valor de x é 2.355506725043957861
Na iteração k = 03, o valor de x é 2.357550606586455544
Na iteração k = 04, o valor de x é 2.357551053877380731
Na iteração k = 05, o valor de x é 2.357551053877402047
Na iteração k = 06, o valor de x é 2.357551053877402047
```

Note que agora as duas raízes podem ser encontradas (aproximadamente) com o Método de Newton, dependendo do chute inicial.

O número de iterações é muito menor do que nos outros métodos, e a convergência é mais rápida. Quanto melhor o chute inicial, menos iterações são necessárias.

4.1 Método da Secante

Em algumas aplicações práticas, f' pode ser de difícil acesso ou até mesmo impossível de calcular, como no caso de resultados experimentais, onde apenas a função f pode ser obtida.

O Método da Secante é uma variação do Método de Newton onde a derivada f' é subtituída por uma aproximação usando diferenças finitas. Essa aproximação é feita com o valor da função em x_k e em x_{k-1} (passo anterior):

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \ .$$

Assim, subtituindo essa aproximação na fórmula para o Método de Newton temos

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \ , \quad k = 1, 2, \cdots$$

Essa é a equação para o **Método da Secante**. Com relação ao Método de Newton, o que muda é só a maneira de calcular a derivada. Abaixo temos o algoritmo do Método da Secante.

Algoritmo: Método da Secante

Entradas: função f; aproximações iniciais x_0 e x_1; tolerância $atol$.

Saída: valor aproximado da solução

1. Comece com as aproximações x_0 e x_1
2. $k \leftarrow 1$
3. $dif \leftarrow 2 \cdot atol$
4. enquanto $dif >= atol$, faça:
    1. $x_{k+1} \leftarrow x_k - (f(x_k)(x_k - x_{k-1}))\ /\ (f(x_k) - f(x_{k-1}))$
    2. $dif \leftarrow |x_{k+1} - x_k|$
    3. $k \leftarrow k + 1$

Fim do Algoritmo

4.2 Convergência

Em uma **sequência covergente**, a **Ordem de Convergência** e a **Taxa de Convergência** são medidas da velocidade com que a sequência converge para o valor final, o seu limite.

Considerando $k \to \infty$, temos

$$|x_{k+1} - x^*| \leq \rho |x_k - x^*|^m \ .$$

Aqui, $\rho < 1$ é a taxa de convergência do método e $m \geq 1$ é a ordem de convergência do método (ou simplesmente ordem do método).

O método possui convergência

1. **linear** se $m = 1$
2. **quadrática** se $m = 2$
3. **cúbica** se $m = 3$

e assim por diante.

O método pode ainda ter convergência **superlinear**. Um método possui convergência superlinear se existe uma sequência $\rho_k \to 0$ tal que, para $k \to \infty$, temos

$$|x_{k+1} - x^*| \leq \rho_k |x_k - x^*| \ .$$

Um método com convergência superlinear converge mais rapidamente do que um método com convergência linear.

Teorema: Convergência do Método de Newton e da Secante. Seja $f(x)$ uma função contínua no intervalo $[a, b]$ com f' e f'' também contínuas no intervalo. Considere que existe uma raiz x^* em $[a, b]$ tal que $f(x^*) = 0$ e $f'(x^*) \neq 0$. Então existe um número $\delta > 0$ tal que, começando com x_0 (e também com x_1, no caso do método da secante) de qualquer lugar no intervalo $[x^* - \delta, x^* + \delta]$, o método de Newton tem convergência quadrática e o método da secante tem convergência superlinear.

4.3 Comentários sobre os Métodos

Alguns erros ou problemas que podem ser encontrados na hora de implementar os métodos apresentados:

1. falta de uma boa aproximação inicial
2. convergência para a raiz errada
3. convergência lenta ou até mesmo não convergência
4. raízes próximas
5. múltiplas raízes

Cada método possui vantagens e desvantagens. O melhor caminho é primeiro tentar fazer o gráfico da função $f(x)$ para tentar identificar algumas de suas propriedades. Depois, seria interessante, por exemplo, utilizar o método da bissecção para reduzir o intervalo de busca. Com o intervalo reduzido e conhecendo bem o comportamento da função, aí sim seria interessante aplicar o método de Newton, já que esse método converge rapidamente, mas em muitos casos é necessário que o chute inicial esteja relativamente próximo da solução desejada para que ele funcione bem.

5 Apêndice

5.1 Série e Polinômio de Taylor

Seja f uma função contínua em $[a, b]$ e com as primeiras k derivadas contínuas no intervalo. Suponha que $f^(k+1)$ exista em $[a, b]$ e que $x_0 \in [a, b]$. Para todo $x \in [a, b]$, existe um número $\xi(x)$ entre x_0 e x tal que

$$f(x) = P_k(x) + R_k(x)$$

onde

$$P_k(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k =$$
$$= \sum_{m=0}^{k} \frac{f^{(m)}(x_0)}{m!}(x - x_0)^m$$

e

$$R_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!}(x - x_0)^{(k+1)} \ .$$

Aqui $P_k(x)$ é chamado de **polinômio de Taylor de ordem k** da função f em torno do ponto x_0 e $R_k(x)$ é chamado de **resto** ou **erro de truncamento** relativo a $P_n(x)$.

A série infinita obtida quando $k \to \infty$ é chamada de **Série de Taylor**.

Na hora de calcular uma aproximação para o valor de uma função, temos truncar a série de Taylor, ou seja, utilizar um polinômio de Taylor de ordem k, desconsiderando o resto:

$$f(x) \approx P_k(x) \ .$$

O erro cometido nessa aproximação é $O(h^{(k+1)})$ (considerando que $f^{(k+1)}(\xi)$ é limitada), ou seja, é de ordem $k + 1$, em que $h = x - x_0$.

5.2 Teorema do Valor Intermediário

Teorema: se $f(x)$ é uma função contínua no intervalo $[a, b]$ e s é um número tal que $f(\hat{a}) \leq s \leq f(\hat{b})$ para dois números $\hat{a}, \hat{b} \in [a, b]$, então existe um número $c \in [a, b]$ tal que $f(c) = s$.

5.3 Teorema do Valor Médio

Teorema: se $f(x)$ é uma função contínua no intervalo $[a, b]$ e diferenciável no intervalo (a, b), então existe um número real $c \in (a, b)$ tal que

$$f'(c) = \frac{f(b) - f(a)}{b - a} \ .$$