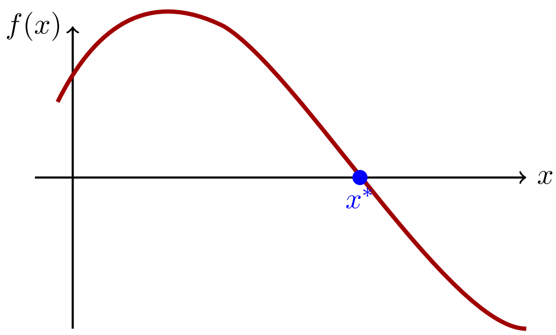
1 Background

In this session we will solve problems of the form

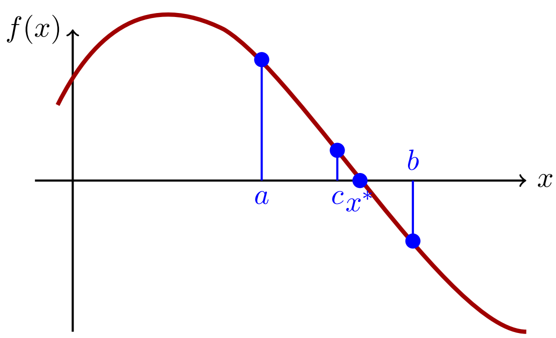
$$f(x) = 0,$$

or equivalently, trying to find the value of $x = x^*$ that makes the function f zero. In this session we will study two possible approaches



1.1 Bissection

Let us consider an interval (a, b) for which we know the solution $x = x^*$ is located



For this example, we notice that $f(a) > 0$ while $f(b) < 0$, this means that in the interval (a, b) the function f changes its sign from positive to negative. Assuming that f is a continuos functio, this also means that somewhere between a and b the function has to go through 0, which is exactly the point we need to find. The bisection algorithm divides the interval (a, b) in two new intervals of the same length, by finding the mid-point

$$c = \frac{a+b}{2}, \tag{1}$$

which for this case has an image larger than zero $f(c) > 0$. Repeating the same argument of before, we then conclude that the solution x^* must be located in the interval (c, b). We can then make the change $a \to c$ and keep narrowing the interval (a, b) until the value of $f(c)$ is small enough.

In order to test this idea we will try to find a root of the equation

$$f(x) = 2x^2 - 3x - 9, \tag{2}$$

in the interval $(0, 5)$. Note that this equation has two solution, but only one of them lies in this interval. The next algorithm shows an implementation of this method

```
def f(x):
    return 2 * x**2 - 3 * x - 9

a = 0.
b = 5.
n = 100

for i in range(n):

    c = 0.5 * (a + b)
    fa = f(a)
    fc = f(c)

    if fa * fc < 0:
        b = c
    else:
        a = c

    if abs(fc) < 1e-3:
        break

    print ('iter = %3d, c = %.8f, f(c) = %.8f') % (i, c, fc)
```

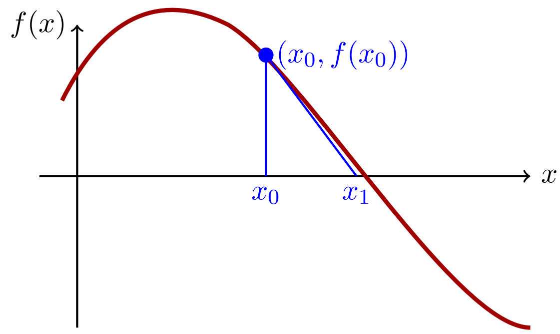
which generates this output

```
iter =   0, c = 2.50000000, f(c) = -4.00000000
iter =   1, c = 3.75000000, f(c) = 7.87500000
iter =   2, c = 3.12500000, f(c) = 1.15625000
iter =   3, c = 2.81250000, f(c) = -1.61718750
iter =   4, c = 2.96875000, f(c) = -0.27929688
iter =   5, c = 3.04687500, f(c) = 0.42626953
iter =   6, c = 3.00781250, f(c) = 0.07043457
iter =   7, c = 2.98828125, f(c) = -0.10519409
iter =   8, c = 2.99804688, f(c) = -0.01757050
iter =   9, c = 3.00292969, f(c) = 0.02638435
iter =  10, c = 3.00048828, f(c) = 0.00439501
iter =  11, c = 2.99926758, f(c) = -0.00659072
iter =  12, c = 2.99987793, f(c) = -0.00109860
iter =  13, c = 3.00018311, f(c) = 0.00164802
```

It is clear that c is indeed approaching the actual solution, and after 13 iterations of the algorithm we have reached a good approximation for the value x^*

1.2 Newton method

The second algorithm we will study in this session is called the Newton method.



Imagine that a point x_0 is given. We can define a straight line that goes through the image of this point, and that is tangent to the plot of f at the point x_0. This equation is simply

$$y - f(x_0) = m(x - x_0), \tag{3}$$

where $m = \mathrm{d}f(x_0)/\mathrm{d}x = f'(x_0)$. Now we can find the location of the point at which this line intersects in the x-axis, if x_1 is such point, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}, \tag{4}$$

which in general yields a better approximation of the solution than x_0.

2 Problems

(P1) How many iterations will it take to find the root of the previous problem with the same accuracy, but using now Newton's method? Start with $x_0 = 5$

```
iter =   0, x = 3.47058824, f(x) = 4.67820069
iter =   1, x = 3.04069952, f(x) = 0.36960861
iter =   2, x = 3.00036156, f(x) = 0.00325430
```

(P2) Define the function

$$f(x) = 1 - \mathrm{e}^{-x}(x + 1) \tag{5}$$

and a function that returns the derivative of $f(x)$

(P3) Set $\mu = 0.1$ and find the value of x for which $f(x) = \mu$ with $\mu = 0.1$

```
iter =   0, x = 0.55354635, f(x) = 0.10685476
iter =   1, x = 0.53200662, f(x) = 0.10006094
iter =   2, x = 0.53181163, f(x) = 0.10000001
```

3 Solutions

(P1)
```
def f(x):
    return 2 * x**2 - 3 * x - 9

def df(x):

    return 4 * x - 3

x = 5.
n = 100

for i in range(n):

    x -= f(x) / df(x)

    if abs(f(x)) < 1e-3:
        break

    print ('iter = %3d, x = %.8f, f(x) = %.8f') % (i, x, f(x))

def f(x, mu):

    return 1 - np.exp(-x) * (x + 1) - mu

def df(x):

    return x * np.exp(-x)
```

(P2)
```
def f(x, mu):

    return 1 - np.exp(-x) * (x + 1) - mu

def df(x):

    return x * np.exp(-x)

mu = 0.1
x = 1.0
n = 100

for i in range(n):

    x -= f(x, mu) / df(x)

    if abs(f(x, mu)) < 1e-10:
        break

    print ('iter = %3d, x = %.8f, f(x) = %.8f') % (i, x, f(x, mu) + mu)
```