Lecture 3

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

The goal of this lecture is to introduce Newton's method and some derivative-free methods to solve nonlinear equations.

0.1 Newton's method (Newton-Raphson method)

Suppose we aim to solve the equation:

$$f(x) = 0$$

Starting from an initial point x_0, the straight line passing through $(x_0, f(x_0))$ tangent to the graph of $f(x)$ is

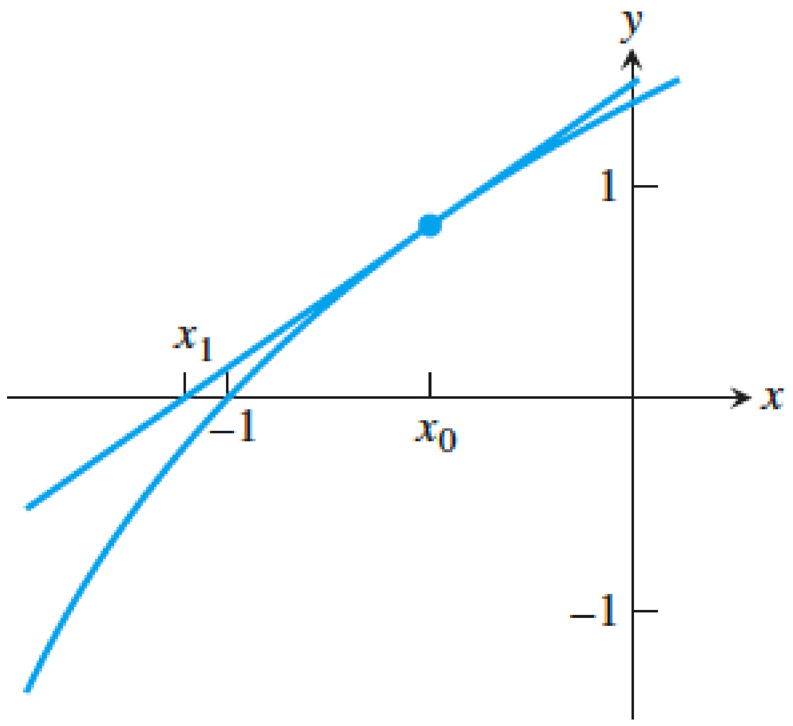
$$y - f(x_0) = f'(x_0)(x - x_0)$$

The line intersects the x-axis at

$$x_0 - \frac{f(x_0)}{f'(x_0)}$$

which is used as the new point x_1 in the iteration.

The one-step Newton process can be illustrated by the following figure:



Repeating the one-step Newton's Method, to obtain $x_2, x_3, \ldots$, yields the following iterative formula:

Newton's Method

$$\begin{aligned} x_0 &= \text{ initial guess} \\ x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \text{ for } i = 0, 1, 2, \ldots \end{aligned}$$

Example 0.1 *Find the Newton's Method formula for the equation $x^3 + x - 1 = 0$.*

Solution

Note

$$f'(x) = 3x^2 + 1$$

The formula is given by

$$x_{i+1} = x_i - \frac{x_i^3 + x_i - 1}{3x_i^2 + 1} = \frac{2x_i^3 + 1}{3x_i^2 + 1}$$

Letting $x_0 = -0.7$, $x_1 \approx 0.1271$, $x_2 \approx 0.9577$.

To show more detail, we further proceed with Newton's method, and show the detail in the following table:

i	x_i	$e_i = \lvert x_i - r \rvert$	e_i / e_{i-1}^2
0	-0.70000000	1.38232780	

i	x_i	$e_i = \lvert x_i - r\rvert$	e_i/e_{i-1}^2
1	0.12712551	0.55520230	0.2906
2	0.95767812	0.27535032	0.8933
3	0.73482779	0.05249999	0.6924
4	0.68459177	0.00226397	0.8214
5	0.68233217	0.00000437	0.8527
6	0.68232780	0.00000000	0.8541
7	0.68232780	0.00000000	

After 6 steps, the root is correct to eight digits. The convergence is quite fast. Actually, Newton's Method is **quadratically convergent** (faster than linear convergence).

0.1.1 Quadratic Convergence of Newton's Method

Definition 0.1 *Let e_i denote the error after step i of an iterative method. The iterations **quadratically convergent** if*

$$M = \lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} < \infty$$

Theorem 0.1 *Let f be twice continuously differentiable and $f(r) = 0$. If $f'(r) \neq 0$, then Newton's Method is locally and quadratically convergent to r. The error e_i at step i satisfies*

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = M$$

where

$$M = \frac{f''(r)}{2f'(r)}$$

Proof 0.1 *Note Newton's Method is a particular form of Fixed-Point Iterations, where*

$$g(x) = x - \frac{f(x)}{f'(x)} \tag{1}$$

The derivative

$$g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2} \tag{2}$$

So $g'(r) = 0$. According to Theorem 2 Lecture 2, Newton's Method is locally convergent.

To show quadratic convergence, we expand f around the current guess x_i with Taylor's formula:

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2} f''(c_i)$$

where c_i is between x_i and x. Evaluate the Taylor expansion at $x = r$:

$$0 = f(r) = f(x_i) + (r - x_i)f'(x_i) + \frac{(r - x_i)^2}{2} f''(c_i)$$

where c_i is between x_i and r. Assuming $f'(x_i) \neq 0$, rearrange the equation:

$$-\frac{f(x_i)}{f'(x_i)} = r - x_i + \frac{(r - x_i)^2}{2} \frac{f''(c_i)}{f'(x_i)}$$

For the next iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = r + \frac{(r-x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)}$$

Equivalently

$$x_{i+1} - r = \frac{(r-x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)}$$

Taking absolute values on both sides:

$$e_{i+1} = e_i^2 \left|\frac{f''(c_i)}{2f'(x_i)}\right|$$

So

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left|\frac{f''(c_i)}{2f'(x_i)}\right| = \left|\frac{f''(r)}{2f'(r)}\right| = M$$

since $c_i \to r$, as $i \to \infty$. So Newton's Method is quadratically convergent.

Return to Example 1. $f'(x) = 3x^2 + 1$ and $f''(x) = 6x$. Then

$$M = \left|\frac{f''(0.6823)}{2f'(0.6823)}\right| \approx 0.85$$

as shown in the previous table.

0.1.2 Linear Convergence of Newton's Method

Note in Theorem 1, we need $f'(r) \neq 0$. This is crucial. We use the following example to show Newton's Method does not converge quadratically, when $f'(r) = 0$.

Example 0.2 *Use Newton's Method to find a root of $f(x) = x^2$.*

Solution

We know there is one root: $r = 0$. Suppose we use the Newton's Method formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^2}{2x_i} = \frac{x_i}{2}$$

i	x_i	$e_i = \lvert x_i - r\rvert$	e_i/e_{i-1}
0	1.000	1.000	
1	0.500	0.500	0.500
2	0.250	0.250	0.500
3	0.125	0.125	0.500
⋮	⋮	⋮	⋮

So Newton's method does converge to the root $r = 0$. The root formula is $e_{i+1} = e_i/2$, so the convergence is linear with convergence proportionality constant $S = 1/2$.

Definition 0.2 *Assume that r is a root of the differentiable function f: that is, assume that $f(r) = 0$. Then if $0 = f(r) = f'(r) = f''(r) = \cdots = f^{(m-1)}(r)$, but $f^{(m)}(r) \neq 0$, we say that f has a **root** of*

multiplicity *m at r. We say that f has a* ***multiple root*** *at r if the multiplicity is greater than one. The root is called* ***simple*** *if the multiplicity is one.*

So there are two diffent rates of convergence to the root r possible in Newton's Method. If $f'(r) \neq 0$ (simple root), the convergence is quadratic, which is faster, if $f'(r) = 0, f''(r) \neq 0$ (multiple root with multiplicity 2), the convergence is linear, which is slower and puts Newton's Method in the same category as bisection and FPI.

In the case when $f'(r) = 0$, there is a theorem that determines the rate S of linear convergence of Newton's method.

Theorem 0.2 *Assume that the $(m+1)$-times continuously differentiable function f on $[a, b]$ has a multiplicity m root at r. Then Newton's Method is locally convergent to r, and the error e_i at step i satisfies*

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = S$$

where $S = (m-1)/m$.

Example 0.3 *Find the multiplicity of the root $r = 0$ of $f(x) = \sin x + x^2 \cos x - x^2 - x$, and estimate the number of steps of Newton's Method required to converge within six correct places (use $x_0 = 1$).*

Solution

Note

$$\begin{aligned} f(x) &= \sin x + x^2 \cos x - x^2 - x \\ f'(x) &= \cos x + 2x \cos x - x^2 \sin x - 2x - 1 \\ f''(x) &= -\sin x + 2\cos x - 4x \sin x - x^2 \cos x - 2 \\ f'''(x) &= -\cos x - 6 \sin x - 6x \cos x + x^2 \sin x \end{aligned}$$

So $f(r) = 0, f'(r) = 0, f''(r) = 0$, but $f'''(r) = -1 \neq 0$. So the multiplicity of $r = 0$ is 3. So Newton's Method converge linearly with $e_{i+1} = \frac{2}{3}e_i$.

For $x_0 = 1$, the initial error $e_0 = 1$. After n steps the error is $e_n = \left(\frac{2}{3}\right)^n e_0$=$\left(\frac{2}{3}\right)^n$. Now we want

$$\left(\frac{2}{3}\right)^n < 0.5 \times 10^{-6}$$

So

$$n > \frac{\log_{10}(.5) - 6}{\log_{10}\left(\frac{2}{3}\right)} \approx 35.78$$

So approximate 36 steps will be needed.

We now implement Newton's Method for Example 3.

[4]:
```python
def Newton(f, fp, x_0, k_max):
    """
    Newton Ralphsom Method
    f: the function
    fp: first derivative of the function
    x_0: initial guess
    k_max: the number of iterations
    """
    res = np.zeros(k_max+1, )
    res[0] = x_0
    for k in range(1, k_max+1):
        res[k] = res[k-1] - f(res[k-1])/fp(res[k-1])

    return res[k_max]
```

[5]:
```
def f_ex3(x):
    return np.sin(x)+x**2*np.cos(x)-x**2-x

def fp_ex3(x):
    return np.cos(x)+2*x*np.cos(x)-x**2*np.sin(x)-2*x-1

k_max = 20
x_0 = 1.
r = 0
res = np.zeros(k_max+1, )
res[0] = x_0

print(' i  ', '        xi  ', '  ei=|xi-r|', '  ei/ei-1 ')
print("{0:<7d} {1:10.7f} {2:10.7f} {3:10s}".format(
        0, x_0, np.abs(x_0-r), '       '))

for k in range(1, k_max+1):
    res[k] = Newton(f_ex3, fp_ex3, x_0, k)
    print("{0:<7d} {1:10.7f} {2:10.7f} {3:7f}".format(
        k, res[k],  np.abs(res[k]-r),
        np.abs(res[k]-r)/np.abs(res[k-1]-r)))
```

```
 i           xi     ei=|xi-r|   ei/ei-1
0        1.0000000  1.0000000
1        0.7215902  0.7215902 0.721590
2        0.5213710  0.5213710 0.722530
3        0.3753083  0.3753083 0.719849
4        0.2683635  0.2683635 0.715048
5        0.1902616  0.1902616 0.708970
6        0.1336125  0.1336125 0.702257
7        0.0929253  0.0929253 0.695483
8        0.0640393  0.0640393 0.689148
9        0.0437781  0.0437781 0.683613
10       0.0297281  0.0297281 0.679063
11       0.0200817  0.0200817 0.675513
12       0.0135121  0.0135121 0.672858
13       0.0090658  0.0090658 0.670938
14       0.0060703  0.0060703 0.669582
```

```
15        0.0040589  0.0040589 0.668642
16        0.0027113  0.0027113 0.667998
17        0.0018100  0.0018100 0.667561
18        0.0012077  0.0012077 0.667266
19        0.0008056  0.0008056 0.667067
20        0.0005373  0.0005373 0.666934
```

The results are as we expected.

If the multiplicity of a root is known in advance, convergence of Newton's Method can be improved with a small modification.

Theorem 0.3 *If f is $(m+1)$-times continuously differentiable on $[a,b]$, which contains a root r of multiplicity $m>1$, then **Modified Newton's Method***

$$x_{i+1} = x_i - \frac{mf(x_i)}{f'(x_i)}$$

converges locally and quadratically to r.

Example 0.4 *Redo Example 3 using the modified Newton's Method.*

[7]:
```
def Newton_mod(f, fp, m, x_0, k_max):
    """
    Modified Newton Ralphsom Method
    f: the function
    fp: first derivative of the function
    m: the multiplicity
    x_0: initial guess
    k_max: the number of iterations
    """
    res = np.zeros(k_max+1, )
    res[0] = x_0
    for k in range(1, k_max+1):
        res[k] = res[k-1] - m*f(res[k-1])/fp(res[k-1])

    return res[k_max]
```

[8]:
```
k_max = 5
x_0 = 1.
r = 0
m = 3
res = np.zeros(k_max+1, )
res[0] = x_0

print(' i      ', '  x_i   ')
for k in range(1, k_max+1):
    res[k] = Newton_mod(f_ex3, fp_ex3, m, x_0, k)
    print("{0:2d} {1:13.10f}".format(k, res[k]))
```

```
 i         x_i
 1   0.1647707196
 2   0.0162073377
 3   0.0002465414
 4   0.0000000607
 5  -0.0000000024
```

Note the solution converges much faster.

We now show an example where Newton's Method fails.

Example 0.5 *Apply Newton's Method to*

$$f(x) = 4x^4 - 6x^2 - \frac{11}{4}$$

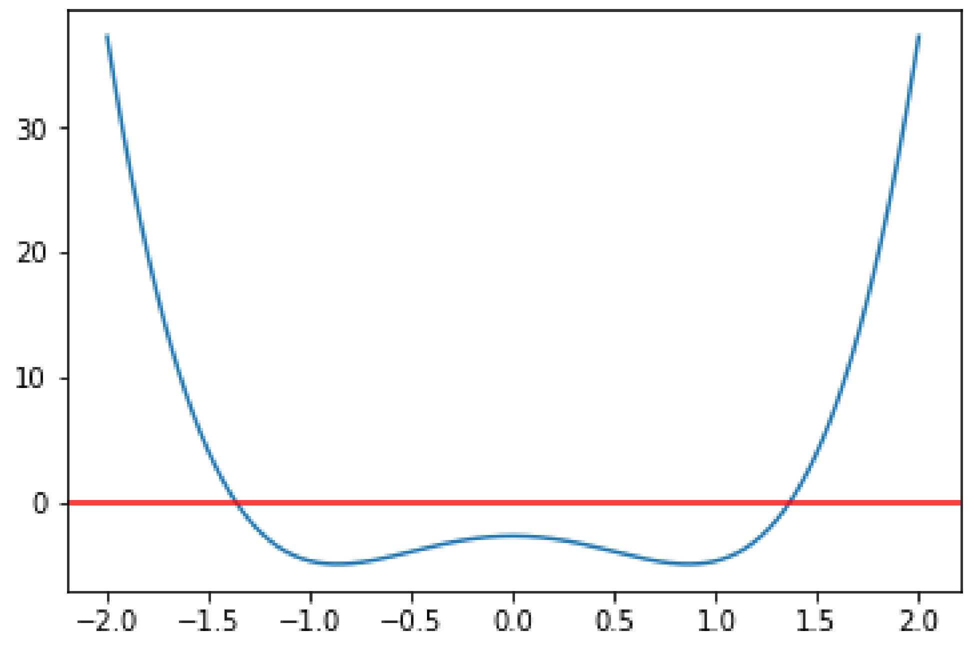
with starting guess $x_0 = \frac{1}{2}$.

We first plot the graph of the function.

[11]:
```
def f_ex5(x):
    return 4*x**4 - 6*x**2 - 11/4

x = np.linspace(-2, 2, 1000)
plt.plot(x, f_ex5(x))
plt.axhline(color='r')
```

[11]:
```
<matplotlib.lines.Line2D at 0x120090ac8>
```



So the solution does exist. Now use Newton's Method.

```
[7]: def fp_ex5(x):
         return 16*x**3 - 12*x

     k_max = 20
     x_0 = .5
     res = np.zeros(k_max+1, )
     res[0] = x_0

     print(' i      ', '  x_i   ')
     for k in range(1, k_max+1):
         res[k] = Newton(f_ex5, fp_ex5, x_0, k)
         print("{0:2d} {1:13.10f}".format(k, res[k]))
```

```
 i         x_i
 1 -0.5000000000
 2  0.5000000000
 3 -0.5000000000
 4  0.5000000000
 5 -0.5000000000
 6  0.5000000000
 7 -0.5000000000
 8  0.5000000000
 9 -0.5000000000
10  0.5000000000
11 -0.5000000000
12  0.5000000000
13 -0.5000000000
14  0.5000000000
15 -0.5000000000
16  0.5000000000
17 -0.5000000000
18  0.5000000000
19 -0.5000000000
20  0.5000000000
```

So the solution oscilates between ± 0.5. Newton's method fails. The initial guess is not sufficiently close to a true solution.

Additional Exercises

Exercise 0.1 *Apply two steps of Newton's Method with initial guess* $x_0 = 0$.

(a) $x^3 + x - 2 = 0$

(b) $x^4 - x^2 + x - 1 = 0$

(c) $x^2 - x - 1 = 0$

Exercise 0.2 *Estimate the error* e_{i+1} *in terms of the previous error* e_i *as Newton's Method converges to the* *given roots. Is the convergence linear or quadratic?*

(a) $x^5 - 2x^4 + 2x^2 - x = 0; r = -1, r = 0, r = 1$

(b) $2x^4 - 5x^3 + 3x^2 + x - 1 = 0; r = -1/2, r = 1$

Exercise 0.3 *Let*

$$f(x) = x^4 - 7x^3 + 18x^2 - 20x + 8$$

Does Newton's Method converge quadratically to the root $r = 2$*? Find* $\lim_{i\to\infty} e_{i+1}/e_i$*, where* e_i *denotes the error at step* i*.*

0.2 Root-Finding Without Derivatives

Newton's Method converges faster than the bisection and FPI methods, because it uses more information—the function's derivative. In some circumstances, the derivative may not be available.

The Secant Method is a good substitute for Newton's Method in this case. It replaces the tangent line with an approximation called the secant line, and converges almost as quickly.

0.2.1 Secant Method and Variants

The Secant Method is similar to the Newton's Method; however, instead of using the tangent line passing through the current point x_i to obtain the next point x_{i+1}, the secant line passing through x_i and x_{i-1} is used.

That is, the derivative of the function, $f'(x_i)$, is replaced by

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

So the iteration formula is now

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

Secant Method

$$\begin{aligned} x_0, x_1 &= \text{ initial guesses} \\ x_{i+1} &= x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \text{ for } i = 1, 2, 3, \ldots \end{aligned}$$

So two starting guesses are needed.

Rate of Convergence

Suppose the Secant Method converges to r and $f'(r) \neq 0$, the approximate error relationship is

$$e_{i+1} \approx \left| \frac{f''(r)}{2f'(r)} \right|^{\alpha+1} e_i^{\alpha}$$

where $\alpha = \frac{(1+\sqrt{5})}{2} \approx 1.62$.

So the convergence of the Secant Method to simple roots is **superlinear** (between linear and quadratic).

Example 0.6 *Apply the Secant Method with starting guesses $x_0 = 0, x_1 = 1$ to find the root of $f(x) = x^3 + x - 1$.*

Solution

The iterative formula is

$$x_{i+1} = x_i - \frac{\left(x_i^3 + x_i - 1\right)\left(x_i - x_{i-1}\right)}{x_i^3 + x_i - \left(x_{i-1}^3 + x_{i-1}\right)}$$

Starting with $x_0 = 0$ and $x_1 = 1$, we have

$$x_2 = 1 - \frac{(1)(1-0)}{1+1-0} = \frac{1}{2}$$
$$x_3 = \frac{1}{2} - \frac{-\frac{3}{8}(1/2-1)}{-\frac{3}{8}-1} = \frac{7}{11}$$

We write a function to perform more steps of Secant Method.

[8]:
```
def secant(f, x_0, x_1, k_max):
    """
    The secant method
    f: the function
    x_0, x_1: the first and second guesses
    k_max: perform k_max steps
    """
    res = np.zeros(k_max+2, )
    res[0] = x_0
    res[1] = x_1
    for k in range(k_max):
        res[k+2] = res[k+1] - (f(res[k+1])*(res[k+1]-res[k]))/
 ↪(f(res[k+1])-f(res[k]))

    return res[k_max+1]
```

[9]:
```
def f_ex6(x):
    return x**3+x-1

x_0 = 0
x_1 = 1
k_max = 8

print(' i   ', '    xi    ')
print("{0:2d} {1:16.14f}".format(0, x_0))
print("{0:2d} {1:16.14f}".format(1, x_1))
for k in range(1, k_max+1):
    res = secant(f_ex6, x_0, x_1, k)
    print("{0:2d} {1:16.14f}".format(k+1, res))
```

```
 i       xi
 0 0.00000000000000
 1 1.00000000000000
 2 0.50000000000000
 3 0.63636363636364
 4 0.69005235602094
 5 0.68202041964819
 6 0.68232578140989
 7 0.68232780435903
 8 0.68232780382802
 9 0.68232780382802
```

0.2.2 Generalization of the Secant Method

The Method of False Position (Regula Falsi) The Method of False Position is similar to the Bisection Method. However, the midpoint is replaced by the intersection of x-axis and the secant line passing through $(a, f(a))$ and $(b, f(b))$.

So the next point c is

$$c = a - \frac{f(a)(a-b)}{f(a)-f(b)} = \frac{bf(a)-af(b)}{f(a)-f(b)}$$

Note this point is guaranteed to be in $[a, b]$.

```
Given initial interval [a,b] such that f(a)f(b)<0
for i=1,2,3,...
    c=(bf(a)-af(b))/(f(a)-f(b))
    if f(c)=0,
        stop
    end
    if f(a)f(c)<0
        b=c
    else
        a=c
    end
end
The final interval [a,b] contains a root.
The approximate root is (bf(a)-af(b))/(f(a)-f(b)).
```

The Method of False Position could be an improvement on both the Bisection Method and the Secant Method. However, the Bisection Method guarantees cutting the uncertainty by 1/2 on each step, while the Method of False Position does not. For some examples, it can converge slower.

Example 0.7 *Apply the Method of False Position on initial interval $[-1, 1]$ to find the root $r = 0$ of $f(x) = x^3 - 2x^2 + \frac{3}{2}x$.*

Solution

Note

$$x_2 = \frac{x_1 f(x_0) - x_0 f(x_1)}{f(x_0) - f(x_1)} = \frac{1(-9/2) - (-1)1/2}{-9/2 - 1/2} = \frac{4}{5}$$

and $f(-1)f(4/5) < 0$, the new interval is $[-1, 0.8]$. So the uncertainty in the solution decreases by only $1/5$. Further steps continue to make slow progress.

Muller's Method Muller's Method is a generalization of Secant Method, where three previous points are used to draw a parabola $y = p(x)$ through them, and the intersection of the parabola with the x-axis is used as the next point.

Note the parabola can intersect the x-axis at 0, 1 or 2 points. If there are two intersections, the one nearest to the most recent point is chosen. If the parabola has no intersection with the x-axis, then complex number solutions are used.

Muller's method can be advantageous, if complex roots are looked for; it can be a bad choice if all roots are real. So it really depends on the problem.

0.2.3 Additional Exercises

Exercise 0.4 *Apply two steps of the Secant Method to*

$$x^3 = 2x + 2$$

with initial guesses $x_0 = 1$ and $x_1 = 2$

Exercise 0.5 *Apply two steps of the Method of False Position with initial bracket $[1, 2]$ to the equation in Exercise 4.*