Review. If $f(x^*)=0$ and $f'(x^*)\neq 0$, then Newton's method (locally) converges to x^* quadratically with rate $\frac{1}{2}|f''(x^*)/f'(x^*)|$.

Note that we can see from here that $f'(x^*)=0$ is problematic; indeed, in that case, we don't get quadratic convergence (but rather divergence or linear convergence).

We can also see that, if $f''(x^*)=0$, then we should get even better convergence; indeed, in that case, we get cubic convergence or better.

Example 65. Consider $f(x)=(x-r)(x-1)(x+2)$ where r is some constant. Suppose we want to use Newton's method to calculate the root $x^*=1$.

(a) For which values of r is Newton's method guaranteed to converge (at least) quadratically to $x^*=1$?

(b) Analyze the cases in which Newton's method does not converge quadratically to $x^*=1$. Does it still converge? If so, what can we say about the order and rate of convergence?

(c) For which values of r does Newton's method converge to $x^*=1$ faster than quadratically?

Solution.

(a) We have $f(x)=x^3-(r-1)x^2-(r+2)x+2r$ and, hence, $f'(x)=3x^2-2(r-1)x-(r+2)$.
Note that $f'(1)=3-3r=0$ if and only if $r=1$.
Theorem 63 implies that Newton's method converges (at least) quadratically to $x^*=1$ if $r\neq 1$.
Comment. Note that $r=1$ is precisely the case where 1 becomes a double root of $f(x)$.

(b) We need to analyze the case $r=1$.
In that case $f(x)=(x-1)^2(x+2)$ and $f'(x)=3x^2-3=3(x-1)(x+1)$.
Newton's method applied to $f(x)$ is equivalent to fixed-point iteration of

$$g(x)=x-\frac{f(x)}{f'(x)}=x-\frac{(x-1)^2(x+2)}{3(x-1)(x+1)}=x-\frac{x^2+x-2}{3(x+1)}=\frac{2}{3}x+\frac{2}{3}\frac{1}{x+1}.$$

We compute that $g'(x)=\frac{2}{3}-\frac{2}{3}\frac{1}{(x+1)^2}$ so that, in particular, $g'(1)=\frac{2}{3}-\frac{2}{3}\frac{1}{4}=\frac{1}{2}$.
Since $0\neq|g'(1)|<1$ we conclude, by Theorem 61, that Newton's method (locally) converges to $x^*=1$. Moreover, the convergence is linear with rate $\frac{1}{2}$.
Comment. Since $\frac{1}{2}=2^{-1}$, this means that we gain roughly one correct binary digit per iteration.

(c) We continue the calculation from the first part. According to Theorem 63, Newton's method converges to 1 faster than quadratic if $f'(1)\neq 0$ and $f''(1)=0$.
We calculate $f''(x)=6x-2(r-1)$. Thus $f''(1)=8-2r=0$ if and only if $r=4$.
Hence, Newton's method converges to 1 faster than quadratic if $r=4$.

Important comment. Note that what we are observing is exactly as what we should expect: Newton's method typically converges quadratically (though in very special cases it can converge even faster; here, $r=4$) except in the case of repeated roots (here, $r=1$).

Armin Straub
straub@southalabama.edu

Example 66. Python The following code implements the Newton method specifically for computing a root of $f(x)=(x-r)(x-1)(x+2)$ as in the previous example.

```
>>> def newton_f(r, x, nr_steps):
        for i in range(nr_steps):
            x = x - ((x-r)*(x-1)*(x+2))/(3*x**2-2*(r-1)*x-r-2)
        return x
```

We then write a function to tell us the how close the result of Newton's method is to $x^*=1$ (the root that we are trying to compute). Namely, `newton_f_cb_1` will return the number of correct digits in base 2.

```
>>> from math import log2

>>> def newton_f_cb_1(r, x, nr_steps):
        return -log2(abs(1 - newton_f(r, x, nr_steps)))
```

Here is the typical behaviour which we get if $r\neq 1$ and $r\neq 4$. We chose $r=2$ and for the initial approximation we chose $x_0=0.4$. First, we list the result of Newton's method and observe that the approximations are indeed approaching 1 (recall that we are only guaranteed convergence if x_0 is close enough to 1). We then list the number of correct bits for those approximations:

```
>>> [newton_f(2, 0.4, n) for n in range(1,5)]

   [0.9333333333333332, 0.9974499089253187, 0.9999956903710115, 0.9999999999876182]

>>> [newton_f_cb_1(2, 0.4, n) for n in range(1,5)]

   [3.9068905956085165, 8.615235511834927, 17.824004894803025, 36.2329923774517]
```

Observe how the number of correct digits indeed roughly doubles.

Next, we likewise consider the problematic case $r=1$:

```
>>> [newton_f(1, 0.4, n) for n in range(1,5)]

   [0.7428571428571429, 0.877751756440281, 0.9402023433223725, 0.9704083354780979]

>>> [newton_f_cb_1(1, 0.4, n) for n in range(1,5)]

   [1.9593580155026542, 3.032114357937968, 4.063767239896592, 5.078665339814252]
```

Observe how the number of correct digits no longer doubles. Instead it roughly increases by 1 per iteration, exactly as we had predicted.

Finally, we consider the exceptionally good case $r=4$:

```
>>> [newton_f(4, 0.4, n) for n in range(1,5)]

   [1.0545454545454547, 0.9999639010889838, 1.0000000000000104, 1.0]

>>> [newton_f_cb_1(4, 0.4, n) for n in range(1,4)]

   [4.1963972128035, 14.757685157968053, 46.445411148322364]
```

Observe how the number of correct digits now roughly triples, in accordance with our prediction.

Comparison of root finding algorithms

Now that we have seen several root finding algorithms, which one is the best?

Well, it really depends on the situation. Below are some of the differences between the methods.

In practice, one often uses hybrid algorithms that combine several methods.

All methods require a continuous function.

- Bisection
  - each iteration is guaranteed to provide a correct binary digit; no other method can guarantee this for all functions
  - requires an initial interval containing a root such that the function values at the endpoints have opposite signs (in particular, does not work for double roots (or any even order roots)); on the other hand, it provides a guaranteed interval containing the root
  - no requirement on $f(x)$ besides continuity; for the other methods, the performance depends on $f(x)$
  - essentially linear convergence with rate $\frac{1}{2}$
- Regula falsi
  - also requires an initial interval containing a root like bisection
  - one endpoint of the interval typically gets stuck
  - rarely used directly, but rather in its improved forms, such as the Illinois method
  - always converges, typically linearly with variable rate
- Illinois method
  - improved version of regula falsi
  - the interval now shrinks to root
  - always converges, typically with order $\sqrt[3]{3} \approx 1.442$
- Secant method
  - only requires an initial approximation
  - only converges if initial approximation is good enough
  - potential numerical issues due to loss of precision in near zero denominator
  - typical order of convergence $\phi = (1+\sqrt{5})/2 \approx 1.618$
- Newton's method
  - similar to secant method
  - requires derivative
  - extends well to other contexts such as approximating functions or power series rather than numbers
  - typical order of convergence 2
  - however, adjusted for two function evaluations ($f(x)$ and $f'(x)$), order of convergence $\sqrt{2} \approx 1.414$