Lecture `root3`: Nonlinear Equations: Newton's Method

September 19, 2022

Summary: Newton's Method for root–finding. Discussion of advantages and disadvantages. Some aspects taken from C. F. Van Loan's textbook; see below.

References: *Numerical Analysis* by T. Sauer and *Introduction to Scientific Computing* by C. F. Van Loan,

We have been attempting to solve equations of the form

$$f(x) = 0 \qquad \text{or equivalently (one possible choice only)} \qquad x = \tilde{f}(x) \equiv f(x) + x. \tag{1}$$

For the first form we used the bisection method subject to the common (but by no means assured) assumption that $f(x)$ changes sign across the root $x^\star$ of interest. For the second form we used fixed–point iteration. Both methods had advantages and disadvantages as discussed. This lecture introduces Newton's Method for solving $f(x) = 0$.

Newton's Method

The idea behind Newton's Method is summed up in Fig. 1. We approximate $f(x)$ by its tangent line at x_i, and then use the root x_{i+1} of the resulting linear function $\ell_i(x) = f(x_i) + f'(x_i)(x - x_i)$ as an approximation to the root $x^\star$ of $f(x)$. Using the point–slope equation of a line, we then have

$$(0 - f(x_i)) = f'(x_i)\,(x_{i+1} - x_i) \qquad \Longrightarrow \qquad x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{2}$$

Here is the algorithm:

1. Start with initial x_0, and the ability to evaluate both $f(x)$ and $f'(x)$. Set $i = 0$.
2. Define the new guess x_{i+1} as in equation (2).
3. Calculate the error, either $|f(x_{i+1})|$ or $|x_{i+1} - x_i|$ (or the maximum over both). If it's less than a tolerance `tol`, then quit. Otherwise, set $i \leftarrow i + 1$ and go back to step 2.

As in the case of fixed–point iteration, we don't really know that this algorithm will work, so we should also include a termination criterion based on a maximum number of iterations. The critical portions of an actual implementation (the MATLAB function `newton.m`) are as follows:

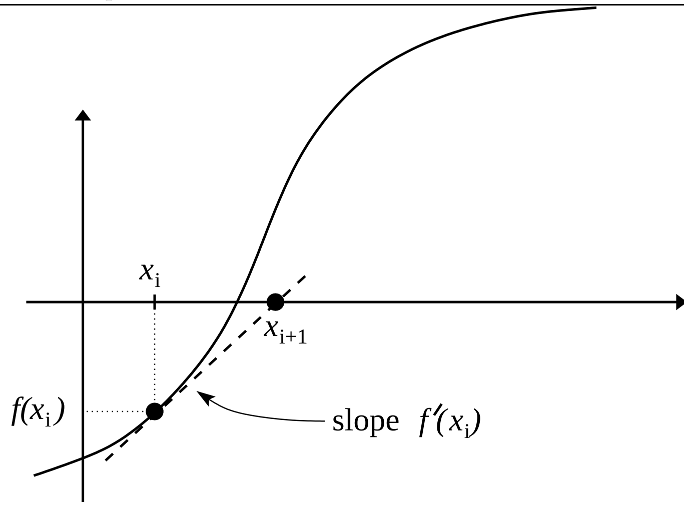

Figure 1: Graphical representation of Newton's Method.

```
  function x = newton(f,df,x0,tol,kmax);
% function x = newton(f,df,x0,tol,kmax);
% Given a differentiable function f with df = df/dx, routine,
% when convergent returns an approximate root x obtained via
% Newton-Raphson iteration. Other inputs are an error tolerance
% tol (max over error between successive iterations and abs(f)),
% the max number kmax allowed iterations, and initial iteration
% x0. tol=1e-8 and kmax = 1e5 are defaults, if left unspecified.
switch nargin
   case 3,
     tol = 1e-8
     kmax = 1e5;
   case 4,
     kmax = 1e5;
   case 5,
     % Fall through.
   otherwise,
     error('newton called with incorrect number of arguments')
end

x = x0; err=100; k = 0;  % x=x0 here allows return for large tol.

while err >= tol
  y = f(x0);
  x = x0 - y/df(x0);
  err = max(abs(y),abs(x-x0));
```

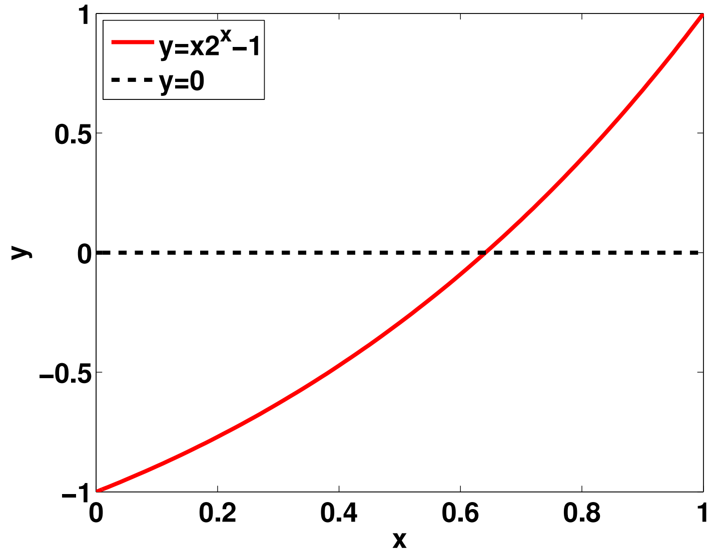

Figure 2: Plot for **Example 1**.

```
  x0 = x;
  k = k+1;
  if k>=kmax
    disp(['No convergence after ' num2str(kmax) ' iterations.'])
    disp(['Error at this stage is ' num2str(err)])
    return
  end
end

if isfinite(x)
  disp(['Converged in ' num2str(k) ' iterations with tol ' num2str(tol)])
else
  disp(['No convergence after ' num2str(k) ' iterations. Iterations' ...
  ' overflowed to infinity.']);
end
```

Note that we require the input `df`, the function that evaluates $f'(x)$. This is one major disadvantage to Newton's method: if the function is complicated, we might not be able to evaluate its derivative.

Example 1

Consider the function and derivative

$$f(x) = x2^x - 1, \qquad f'(x) = 2^x + x2^x \ln(2).$$

As you can see from Fig. 2, there is a single root on $(0, 1)$. Here is the output from `newton`

```
>> format long g; format compact
>> x0 = 1; tol = 1e-10; kmax = 100;
>> newton(@lecroot3_fun1,@lecroot3_dfun1,x0,tol,kmax)
Converged in 5 iterations with tol 1e-10
ans =
         0.641185744504986
>> fzero('lecroot3_fun1',1)
ans =
         0.641185744504986
```

Newton's method works well for this example. In fact, the convergence is *quadratic*, as we'll see later.

Example 2
Just like other methods we've explored for finding roots, Newton's method doesn't necessarily converge to the correct root if the function is unusual. Consider finding the root of

$$f(x) = \frac{1}{2} - \frac{x}{|x|^{1.1} + \frac{1}{300}},$$

with derivative

$$f'(x) = \frac{0.1|x|^{1.1} - \frac{1}{300}}{\left(|x|^{1.1} + \frac{1}{300}\right)^2}.$$

Note, $f(x)$ is obviously differentiable at all points save $x = 0$. To verify that indeed $f'(0) = -300$, you'll have to use the limit definition of the derivative. This function has two roots, one around $x = 0$, and another around $x = 1000$. A plot of the function is given in Fig. 3.

Let's use these expressions with Newton's method to find the root near $x = 0$. We define functions `lecroot3_fun2.m` and `lecroot3_dfun2.m` to evaluate $f(x)$ and $f'(x)$, respectively. If we start the Newton iteration with $x_0 = 0.01$, very close to the root, we converge rather well:

```
>> format long g; format compact
>> x0 = 0; tol = 1e-8; kmax = 100;
>> newton(@lecroot3_fun2,@lecroot3_dfun2,x0,tol,kmax)
Converged in 5 iterations with tol 1e-08
ans =
        0.0022901407314073
```

However, we now show that in some cases Newton's method can give either a completely wrong answer (convergence to a different root), or can diverge (shoot off to infinity).

```
>> format long g; format compact
>> x0 = 0.05; tol = 1e-8; kmax = 100;
>> newton(@lecroot3_fun2,@lecroot3_dfun2,x0,tol,kmax)
```

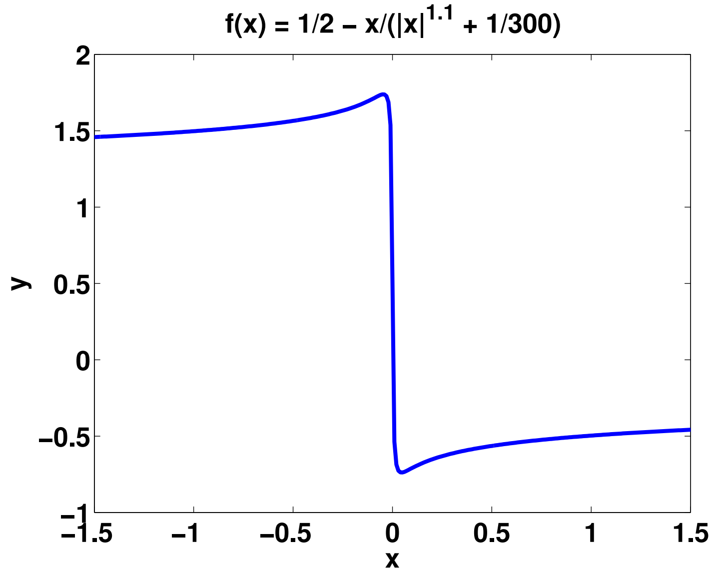

Figure 3: Plot for **Example 2**.

```
Converged in 11 iterations with tol 1e-08
ans =
          1023.98333318413
>> x0 = -0.05;
>> newton(@lecroot3_fun2,@lecroot3_dfun2,x0,tol,kmax)
No convergence after 32 iterations. Iterations overflowed to infinity.
ans =
   NaN
```

In fact, the `while` loop exited because `err = NaN` at the next to last iterate (and MATLAB decided thus to quit the while loop)! In the first case above we took $x_0 = 0.05$, very close to the root near $x = 0$, and the iteration fell into the well of attraction for the root near $x = 1000$ instead. Thus, if we choose an initial condition even relatively close to a root, Newton's method might take us to a different one very far away. In the second case, we took the initial condition to be $x_0 = -0.05$, on the other side of $x = 0$, but still very close to that root. In this case the iteration diverged off to infinity; we didn't even find a root.

Quadratic convergence

In Section 1.4 of the text Sauer explains that near a root $x^\star$ for which $|f'(x^\star)| \neq 0$, Newton's method is *quadratically convergent*. In terms of $e_k = |x_k - x^\star|$, the error between the kth iterate x_k and the exact root $x^\star$, this statement implies that (for x_k sufficiently close to the root)

$$e_{k+1} \simeq M e_k^2, \qquad M = \left| \frac{f''(x^\star)}{2f'(x^\star)} \right|.$$

Notice that the new error e_{k+1} is proportional to the *square* of the old error e_k (that's where *quadratic* comes in). Let's investigate whether the convergence in **Example 1** is indeed quadratic, and we'll start off by getting an estimate for M. We don't know the root, so let's use MATLAB's value as the "exact answer."

```
>> format long g; format compact
>> fzero('x.*2.^x-1',1)
fzero('x.*2.^x-1',1)
ans =
        0.641185744504986
```

Therefore, we find $M \simeq \left|\frac{f''(0.641185744504986)}{2f'(0.641185744504986)}\right| = 0.586510540370655$, where we've done some calculations you'll need to do for the next homework. Now calling `newton` with `tol = 1e-10` and `2.5` as the initial guess, we then find the following iterates (x_0 through x_6).

```
     0                      2.5
     1        1.649895514156352
     2        1.028874989972393
     3        0.714378823893981
     4        0.644211982933515
     5        0.641191107367823
     6        0.641185744521854
```

These x_k correspond to the following errors e_k (here relative to `0.641185744504986` from MATLAB as the "exact answer").

```
     1.858814255495014
     1.008709769651366
     0.387689245467407
     0.073193079388995
     0.003026238428529
     0.000005362862837
     0.000000000016868
```

Finally, we tabulate the ratios e_{k+1}/e_k^2 for $k = 0, 1, 2, 3, 4, 5$. Here they are

```
     0.291940426500243
     0.381023094256533
     0.486970341414769
     0.564888981130286
     0.585585623128099
     0.586509000666890
```

The point is that the final values are indeed close to the value we computed for M. In anycase, the fact that the ratios appear to settle on a fixed number (whether $\simeq 0.59$ or another) indicates quadratic convergence.