Newton-Raphson

The Newton-Raphson method is an approach for finding the roots of nonlinear equations and is one of the most common root-finding algorithms due to its relative simplicity and speed. The root of a function is the point at which $f(x) = 0$ f(x)=0. Many equations have more than one root. Every real polynomial of odd degree has an odd number of real roots ("Zero of a function," 2016). Newton-Raphson is an iterative method that begins with an initial guess of the root. The method uses the derivative of the function $f'(x)$ f′(x) as well as the original function $f(x)$ f(x), and thus only works when the derivative can be determined.

Newton-Raphson Iteration

The initial guess of the root is typically denoted x_0x0 with the true root represented by rr. The true root can thus be expressed as $r = x_0 + h$ r=x0+h, and therefore $h = r - x_0$ h=r−x0, where hh measures how far the guess is from the true value of the root. As hh will be small, a linear tangent line is used to approximate the location of the root and can be written as:

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0)$$
0=f(r)=f(x0+h)≈f(x0)+hf′(x0)

where hh is approximately:

$$h \approx -\frac{f(x_0)}{f'(x_0)}$$
h≈−f(x0)f′(x0)

Unless the derivative $f'(x_o)$f′(xo) is close to 0. Combining this approximation with the value of the true root yields:

$$r = x_0 + h \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$
r=x0+h≈x0−f(x0)f′(x0)

Therefore the new estimate of rr, x_1x1, becomes:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$
x1=x0−f(x0)f′(x0)

Then the new estimate x_2x2 is obtained from x_1x1 in the same manner:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$
x2=x1−f(x1)f′(x1)

The iteration of Newton-Raphson can thus be generalized:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$
xn+1=xn−f(xn)f′(xn)

Newton-Raphson Convergence

Newton-Raphson is not a foolproof method in that it can fail to converge to a solution. In fact, there are no 'perfect' numerical methods that will always converge on a solution. Particularly, the assumption $f''(x)$ f″(x) exists and is continuous near r r must be made.

The method can also fail to converge when $f'(x)$ f′(x) is close to 0. For example, 5.02 is close to 5 as it is only 0.4% different, however, $5.02/10^{-7}$ 5.02/10−7, is quite different than $5/10^{-7}$ 5/10−7 as $(5.02/10^{-7} - 5/10^{-7})/5/10^{-7}$ (5.02/10−7−5/10−7)/5/10−7 is -700% different.

Examples of Newton-Raphson

Before proceeding to an implementation of the Newton-Raphson method in R, it is worth working through some examples to get an understanding of the definitions and equations above. The NR method can be used to approximate square roots such as $\sqrt{10}$ 10. The square root of 10 is about three, so we can use that as a good starting value.

It often helps to plot the function to see where the roots occur. The function is first rearranged to be an expression of $f(x)$ f(x) before plotting.

$$x = \sqrt{10}$$
x=10

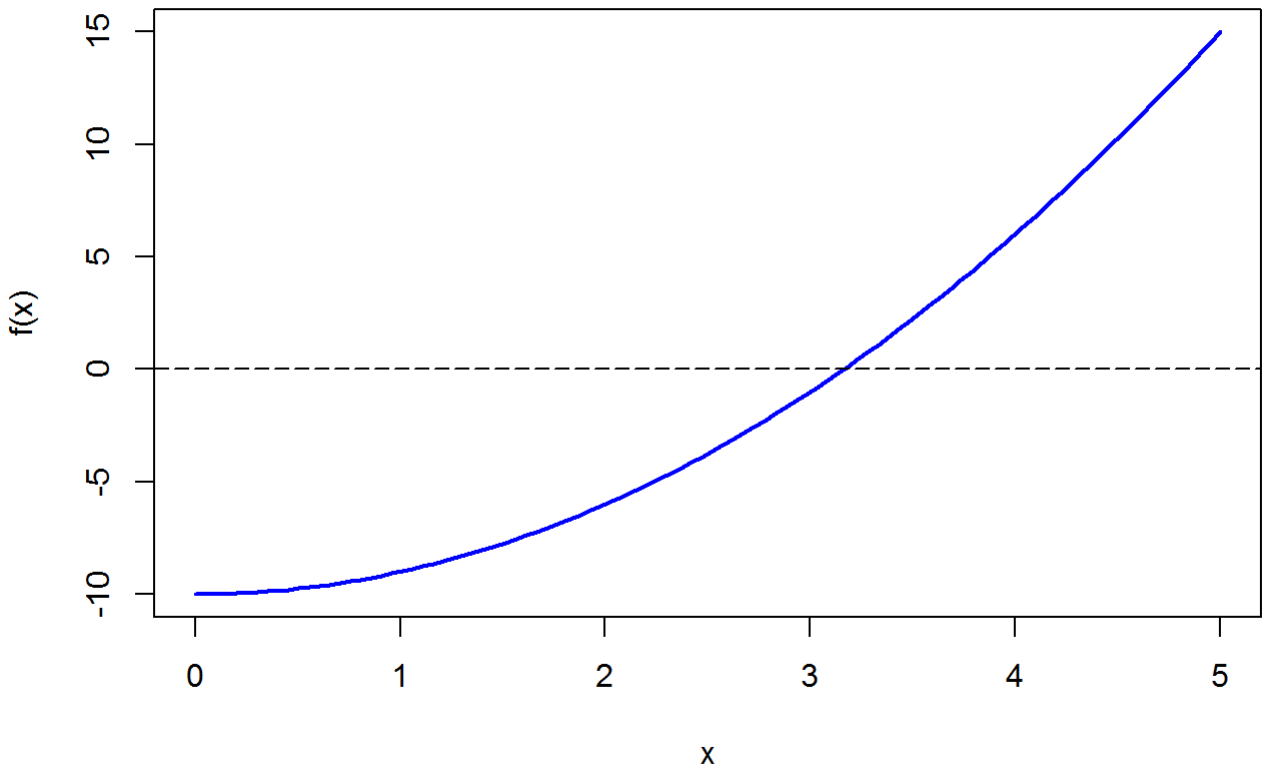
$$x - \sqrt{10} = 0$$
x−10=0

$$f(x) = x^2 - 10$$
f(x)=x2−10

Plotting the function yields the graph below.

```
func <- function(x) {
  x^2 - 10
}

curve(func, col = 'blue', lwd = 2, from = 0, n = 100, xlim=c(0, 5), ylab='f(x)')
abline(a=0, b=0, lty = 5)
```



It can be seen from the graph the function crosses the x-axis at $\sqrt{10}$10 on an interval [3, 4]. The derivative of the function is $2x$ 2x.

Solving for the function root using the Newton-Raphson method proceeds as follows using three as an initial guess.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - 10}{2x_n}$$

xn+1=xn−f(xn)f′(xn)=xn−xn2−102xn

$$x_1 = 3 - \frac{(3)^2 - 10}{2(3)} = 3.166667$$

x1=3−(3)2−102(3)=3.166667

$$x_2 = 3.166667 - \frac{(3.166667)^2 - 10}{2(3.166667)} = 3.162281$$

x2=3.166667−(3.166667)2−102(3.166667)=3.162281

$$x_3 = 3.162281 - \frac{(3.162281)^2 - 10}{2(3.162281)} = 3.162278$$

x3=3.162281−(3.162281)2−102(3.162281)=3.162278

And so on until the x_n xn estimates are within a particular level of tolerance. This example converges in three iterations. Thus $3.1622783.162278$ is the estimated root of the function. This result can be verified by simply taking the square root of 10.

```
sqrt(10)
```

```
## [1] 3.162278
```

Newton-Raphson Method in R

The `uniroot` function in R provides an implementation of Newton-Raphson for finding the root of an equation. The function is only capable of finding one root in the given interval. The rootSolve package features the `uniroot.all` function which extends the `uniroot` routine to detect multiple roots should they exist.

The equation for which we wish to find the root is:

$$f(x) = x^3 - 2x - 5$$

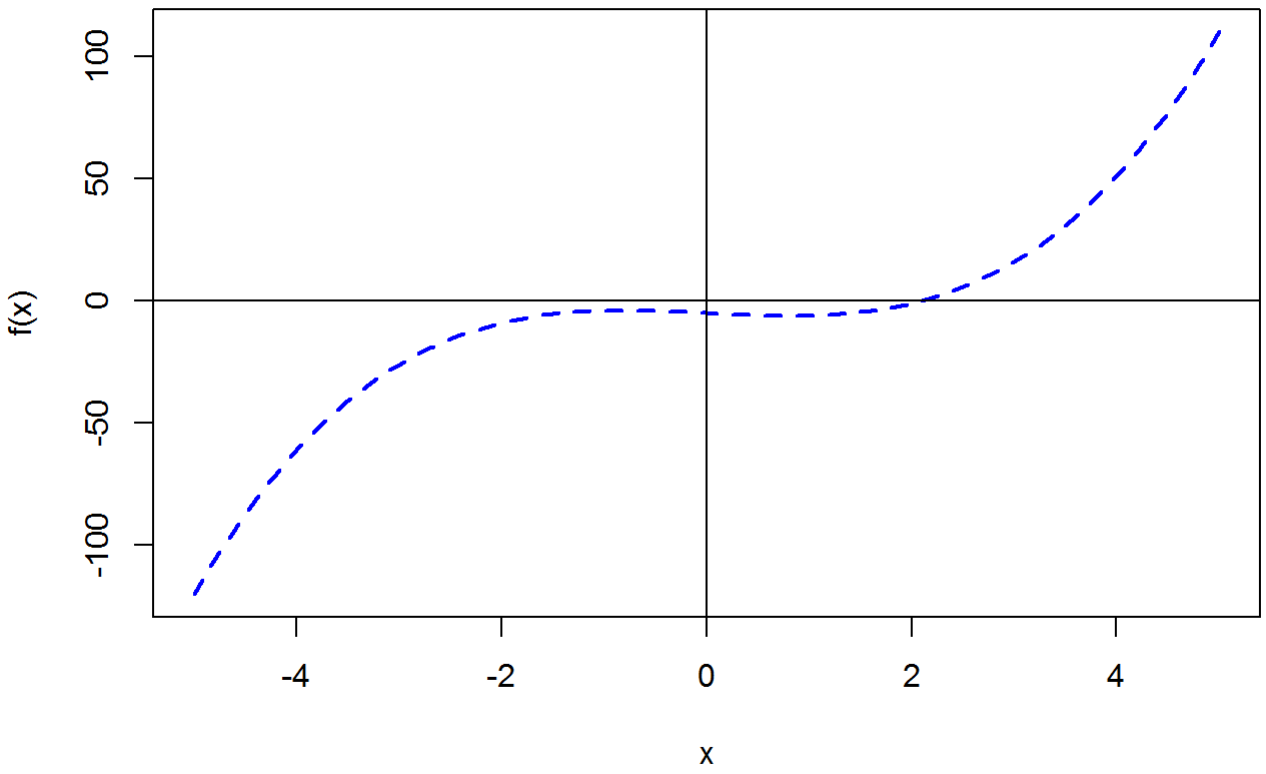
f(x)=x3−2x−5

First, write a function to express the equation above.

```
func2 <- function(x) {
  x^3 - 2* x - 5
}
```

Plot the function to visualize how the equation behaves and where any roots may be located.

```
curve(func2, xlim=c(-5,5), col='blue', lwd=2, lty=2, ylab='f(x)')
abline(h=0)
abline(v=0)
```



It looks like the function equals 0 when yy is about 2. To find the root of the equation, use the `uniroot` function with a starting value of 2 and upper bound of 3.

```
uniroot(func2, c(2,3))
```

```
## $root
## [1] 2.094541
##
## $f.root
## [1] -0.0001147009
##
## $iter
## [1] 5
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

As suspected, the root of the function is very close to 2 at 2.0945412. The iterations can be written as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^3 - 2x - 5}{3x_n^2 - 2}$$

xn+1=xn−f(xn)f′(xn)=xn−xn3−2x−53xn2−2

$$x_1 = 2 - \frac{(2)^3 - 2(2) - 5}{3(2)^2 - 2} = 1.281239$$

x1=2−(2)3−2(2)−53(2)2−2=1.281239

$$x_2 = 1.281239 - \frac{(1.281239)^3 - 2(1.281239) - 5}{3(1.281239)^2 - 2} = 3.147821$$

x2=1.281239−(1.281239)3−2(1.281239)−53(1.281239)2−2=3.147821

$$x_3 = 3.147821 - \frac{(3.147821)^3 - 2(3.147821) - 5}{3(3.147821)^2 - 2} = 2.430257$$

x3=3.147821−(3.147821)3−2(3.147821)−53(3.147821)2−2=2.430257

$$x_4 = 2.430257 - \frac{(2.430257)^3 - 2(2.430257) - 5}{3(2.430257)^2 - 2} = 2.144418$$

x4=2.430257−(2.430257)3−2(2.430257)−53(2.430257)2−2=2.144418

$$x_5 = 2.144418 - \frac{(2.144418)^3 - 2(2.144418) - 5}{3(2.144418)^2 - 2} = 2.095897$$

x5=2.144418−(2.144418)3−2(2.144418)−53(2.144418)2−2=2.095897

$$x_6 = 2.095897 - \frac{(2.095897)^3 - 2(2.095897) - 5}{3(2.095897)^2 - 2} = 2.094552$$

x6=2.095897−(2.095897)3−2(2.095897)−53(2.095897)2−2=2.094552

The manual calculations equal the output of the function to four decimals. The minuscule difference between results is likely the cause of roundoff and significance errors in the manual calculation. A quick and dirty function to perform the method in R can be implemented to further verify our understanding of the Newton-Raphson method. The numDeriv package is used to compute the derivative $f'(x)$ f′(x), though this could also be done by taking the limit with a sufficiently small hh.

```r
newton.raphson <- function(f, a, b, tol = 1e-5, n = 1000) {
  require(numDeriv) # Package for computing f'(x)

  x0 <- a # Set start value to supplied lower bound
  k <- n # Initialize for iteration results

  # Check the upper and lower bounds to see if approximations result in 0
  fa <- f(a)
  if (fa == 0.0) {
    return(a)
  }

  fb <- f(b)
  if (fb == 0.0) {
    return(b)
  }

  for (i in 1:n) {
    dx <- genD(func = f, x = x0)$D[1] # First-order derivative f'(x0)
    x1 <- x0 - (f(x0) / dx) # Calculate next value x1
    k[i] <- x1 # Store x1
    # Once the difference between x0 and x1 becomes sufficiently small, output the results.
    if (abs(x1 - x0) < tol) {
      root.approx <- tail(k, n=1)
      res <- list('root approximation' = root.approx, 'iterations' = k)
      return(res)
    }
    # If Newton-Raphson has not yet reached convergence set x1 as x0 and continue
    x0 <- x1
  }
  print('Too many iterations in method')
}
```

Computing the root of the above equation with the `newton.raphson` function yields:

```r
newton.raphson(func2, 2, 3)
```

```
## Loading required package: numDeriv
```

```
## $`root approximation`
## [1] 2.094551
##
## $iterations
## [1] 2.100000 2.094568 2.094551 2.094551
```

The custom Newton-Raphson function is slightly off from the `uniroot` result, however; they are equal up to 4 decimals and is adequate for our purposes.

One More Example

Let's say we are interested in finding the root of the function:

$$e^{2x} = x + 6$$
e2x=x+6

Setting the equation equal to 0:

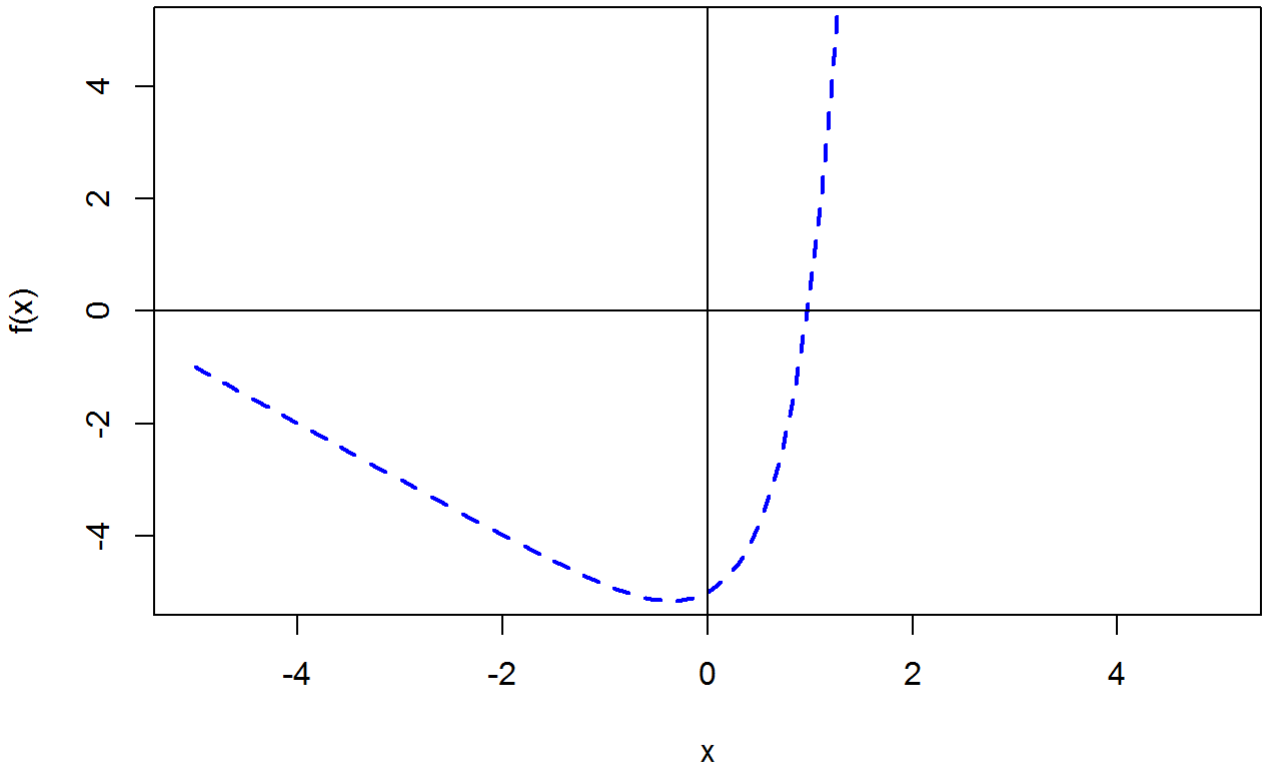
$$f(x) = e^{2x} - x - 6 = 0$$
f(x)=e2x−x−6=0

As before, write the equation as an R function.

```
func3 <- function(x) {
  exp(2 * x) - x - 6
}
```

Plot the function to visualize where the roots may exist.

```
curve(func3, col = 'blue', lty = 2, lwd = 2, xlim=c(-5,5), ylim=c(-5,5), ylab='f(x)')
abline(h=0)
abline(v=0)
```



It appears the function has a root around $f(x) = 1$ f(x)=1. Bounds of $1/2$ 1/2 and $3/2$ 3/2 will be used.

```
uniroot(func3, c(.5, 1.5))
```

```
## $root
## [1] 0.9708565
##
## $f.root
## [1] -0.000175188
##
## $iter
## [1] 6
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

```
newton.raphson(func3, .5, 1.5)
```

```
## $`root approximation`
## [1] 0.97087
##
## $iterations
## [1] 1.3523980 1.0894844 0.9846706 0.9710729 0.9708701 0.9708700
```

The root is located at .9708 as reported by the `uniroot` function and our custom method.

Summary

The Newton-Raphson method is a powerful and relatively straightforward method for finding the roots of an equation. It has many advantages but suffers from several drawbacks such as a readily calculated derivative, inconsistencies when $f'(x) = 0$ f′(x)=0 and so on. As explored in the post, it often makes sense first to plot the function in question to visualize how it behaves before attempting to locate the root(s) as bad starting values can be detrimental to results. Other numerical methods for estimating roots of equations such as the Bisection, Secant and Brent's methods will be examined in future posts.

References

Agresti, A. (2002). Categorical data analysis (2nd ed.). New York, NY: Wiley-Interscience.

Kiusalaas, J. (2013). Numerical methods in engineering with python (2nd ed.). New York: Cambridge University Press.

The Newton-Raphson method. Retrieved from https://www.math.ubc.ca/~anstee/math104/104newtonmethod.pdf

Zero of a function (2016). In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Zero_of_a_function#Polynomial_roots