Bisection method, Derivative Approximation, Optimization

Stats 102A

Miles Chen

Department of Statistics

Week 7 Friday

UCLA

Copyright Miles Chen. For personal use only. Do not distribute.

Section 1

Bisection Method

Copyright Miles Chen. For personal use only. Do not distribute.

Bisection Method

The **bisection method** is one of the simplest numerical methods for root-finding to implement, and it almost always works. The main disadvantage is that convergence is relatively slow.

The idea behind the bisection method is **root-bracketing**, which works by first finding an interval in which the root must lie, and then successively refining the bounding interval in such a way that the root is guaranteed to always lie inside the interval.

In the bisection method, the width of the bounding interval is successively halved.

Copyright Miles Chen. For personal use only. Do not distribute.

Bisection Method

Suppose that f is a continuous function. Then f has a root in the interval (x_ℓ, x_r) if either:

- $f(x_\ell) < 0$ and $f(x_r) > 0$, or
- $f(x_\ell) > 0$ and $f(x_r) < 0$.

A convenient way to verify this condition is to check if

$$f(x_\ell)f(x_r) < 0$$

That is to say, if $f(x_\ell)$ and $f(x_r)$ have opposite signs, then a root must exist between x_ℓ and x_r

The bisection method works by taking an interval (x_ℓ, x_r) that contains a root, then successively refining x_ℓ and x_r until $x_r - x_\ell \leq \varepsilon$, where $\varepsilon > 0$ is the pre-specified tolerance level.

Copyright Miles Chen. For personal use only. Do not distribute.

Bisection Method

The **bisection** algorithm is described as follows:

Start with $x_\ell < x_r$ such that $f(x_\ell)f(x_r) < 0$.

1. If $x_r - x_\ell \le \varepsilon$, then stop.
2. Compute $x_m = \dfrac{x_\ell + x_r}{2}$. If $f(x_m) = 0$, then stop.
3. If $f(x_\ell)f(x_m) < 0$, then assign $x_r = x_m$.
   Otherwise, assign $x_\ell = x_m$.
4. Go back to step 1.

Notice that, at every iteration, the interval (x_ℓ, x_r) always contains a root. Assuming we start with $f(x_\ell)f(x_r) < 0$, the algorithm is guaranteed to converge, with the approximation error reducing by a constant factor $1/2$ at each iteration. If we stop when $x_r - x_\ell \le \varepsilon$, then we know that both x_ℓ and x_r are within ε distance of a root.

Copyright Miles Chen. For personal use only. Do not distribute.

Example of Bisection Method

An implementation of bisection method for $f(x) = \log(x) - e^{-x}$:

```r
f <- function(x) {
  log(x) - exp(-x)
}
```

```r
x_l <- 1
x_r <- 2
tol <- 1e-8
f_l <- f(x_l)
f_r <- f(x_r)
f_l * f_r < 0
```

```
## [1] TRUE
```

Copyright Miles Chen. For personal use only. Do not distribute.

Example of Bisection Method

```r
iterations <- 0
while (x_r - x_l > tol) {
  x_m <- (x_l + x_r) / 2
  f_m <- f(x_m)
  if (identical(all.equal(f_m, 0), TRUE)) {
    break
  }
  if (f_l * f_m < 0) {
    x_r <- x_m
  } else {
    x_l <- x_m
  }
  iterations <- iterations + 1
}
(x_l + x_r) / 2
```

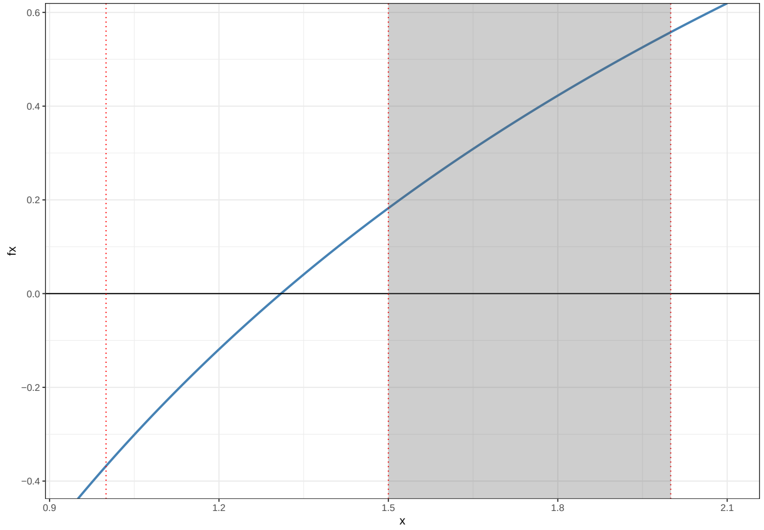
```
## [1] 1.3098
```

```r
iterations
```

```
## [1] 24
```

Copyright Miles Chen. For personal use only. Do not distribute.

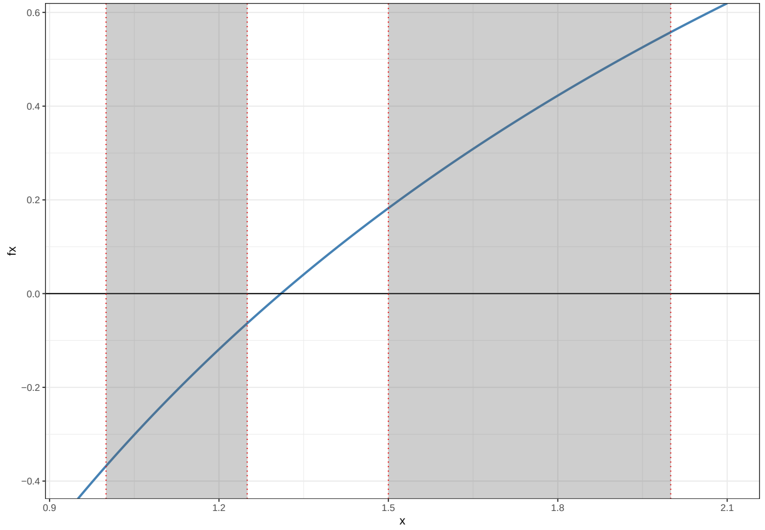
f(x) = log(x) - exp(-x)



```
## [1] 1.5
```

Copyright Miles Chen. For personal use only. Do not distribute.

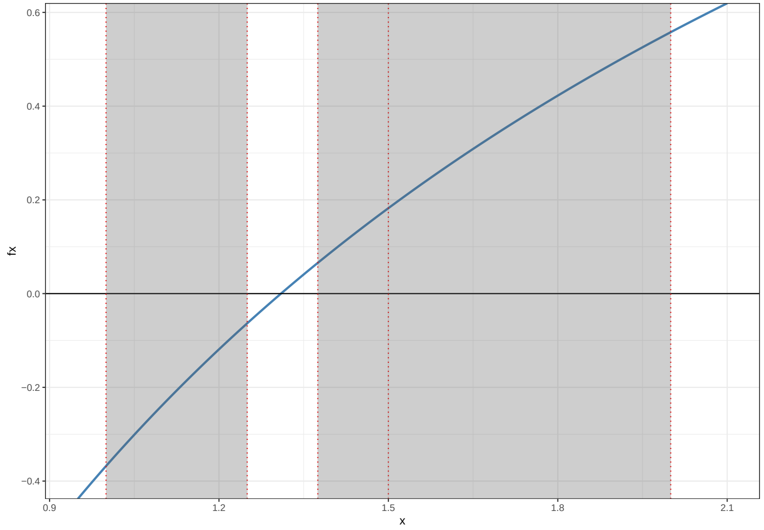
f(x) = log(x) - exp(-x)



```
## [1] 1.25
```

Copyright Miles Chen. For personal use only. Do not distribute.

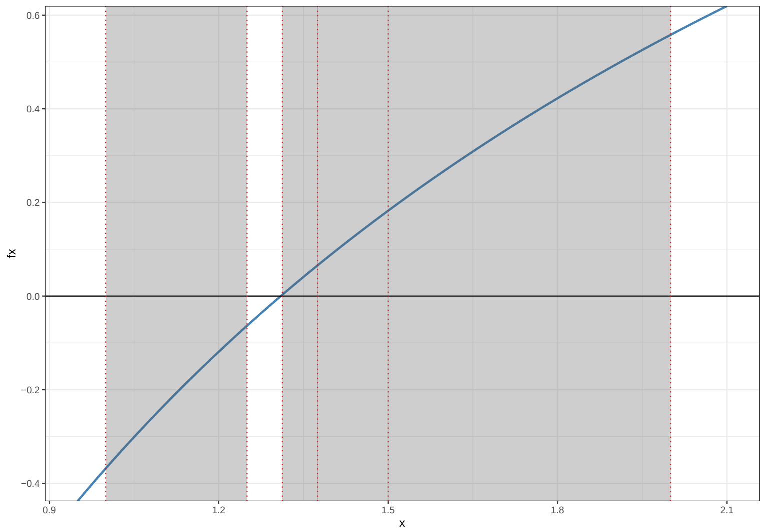
f(x) = log(x) - exp(-x)



```
## [1] 1.375
```

Copyright Miles Chen. For personal use only. Do not distribute.

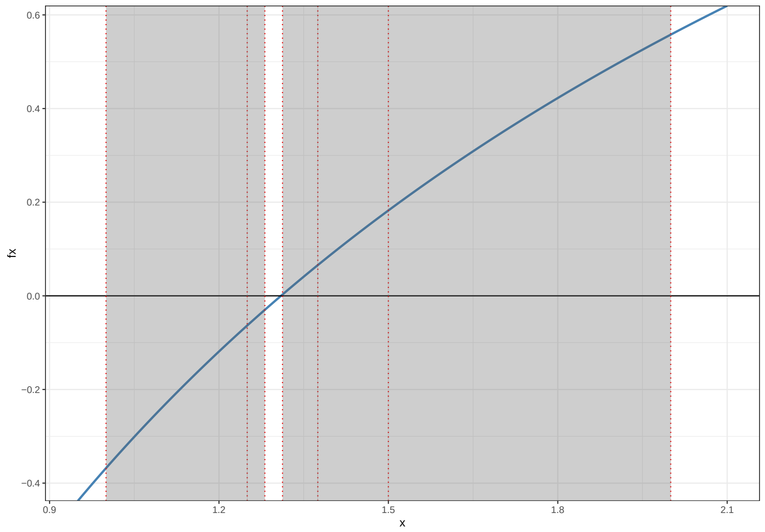
f(x) = log(x) - exp(-x)



```
## [1] 1.3125
```

Copyright Miles Chen. For personal use only. Do not distribute.

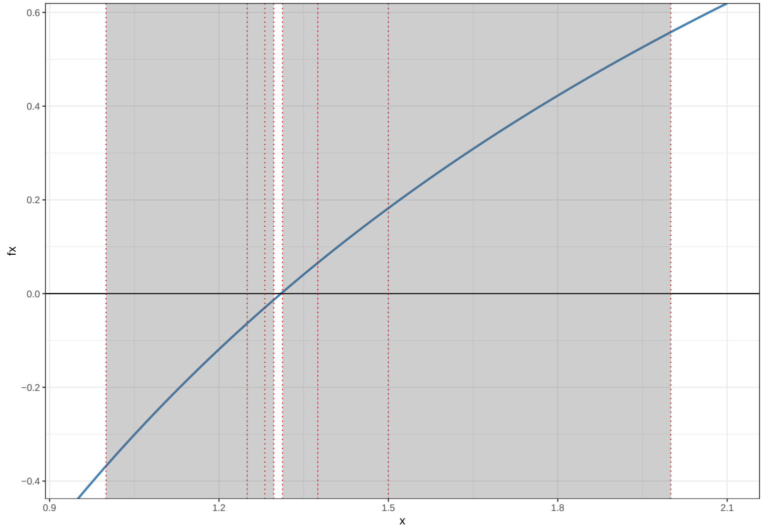
f(x) = log(x) - exp(-x)



```
## [1] 1.28125
```

Copyright Miles Chen. For personal use only. Do not distribute.

f(x) = log(x) - exp(-x)



```
## [1] 1.296875
```

Copyright Miles Chen. For personal use only. Do not distribute.

Best of Both Worlds

The most popular current root-finding methods use root-bracketing to get close to a root, then switch over to the Newton-Raphson or secant method when it seems safe to do so.

This strategy combines the safety of bisection with the speed of the secant method.

Copyright Miles Chen. For personal use only. Do not distribute.

Order of Convergence

Let a be the root and x_n be the nth approximation to the root. Define the **error** to be

$$\varepsilon_n = a - x_n.$$

A root finding method has an **order of convergence** p if, for large n, we have the approximate relationship

$$|\varepsilon_{n+1}| = k|\varepsilon_n|^p,$$

for some positive constant k. Larger values of p correspond to faster convergence.

Side Note: The derivations for the orders of convergence discussed here are in Jeffrey R. Chasnov's lecture notes on numerical methods: https://www.math.ust.hk/~machas/numerical-methods.pdf

Copyright Miles Chen. For personal use only. Do not distribute.

Order of Convergence

The order of convergence of the bisection method is 1, since the error is reduced by approximately a factor of 2 with each iteration. That is,

$$|\varepsilon_{n+1}| = \frac{1}{2}|\varepsilon_n|.$$

Order of convergence 1 is also called **linear convergence**.

It can be shown that, under certain conditions, the Newton-Raphson method has order of convergence 2, i.e.,

$$|\varepsilon_{n+1}| = k|\varepsilon_n|^2,$$

which is also called **quadratic convergence**.

The secant method can be shown to have an order of convergence equal to

$$p = \frac{1+\sqrt{5}}{2} \approx 1.618034,$$

which is the famous **golden ratio** (sometimes denoted by Φ)!

Copyright Miles Chen. For personal use only. Do not distribute.

Section 2

Derivative Approximation

Copyright Miles Chen. For personal use only. Do not distribute.

Newton-Raphson Method

The **Newton-Raphson** algorithm is described as follows:

- Start from an initial value x_0.
- Repeat:

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

Stop when $|f(x_n)| \leq \varepsilon$, where $\varepsilon > 0$ is the pre-specified tolerance level.

Copyright Miles Chen. For personal use only. Do not distribute.

Newton-Raphson Method

The Newton-Raphson Method requires finding the derivative at the point x_n.

When implementing the Newton-Raphson method, we might choose to write a function that calculates the derivative.

Copyright Miles Chen. For personal use only. Do not distribute.

Example

Suppose we want to use the Newton-Raphson method to find the root of

$$f(x) = \log(x) - e^{-x}$$

Taking the derivative with respect to x, we have

$$f'(x) = \frac{1}{x} + e^{-x}$$

Copyright Miles Chen. For personal use only. Do not distribute.

Example

In R, we can write:

```r
f <- function(x) {
  log(x) - exp(-x)
}
fprime <- function(x) {
  (1 / x) + exp(-x)
}
f(5)
```

```
## [1] 1.6027
```

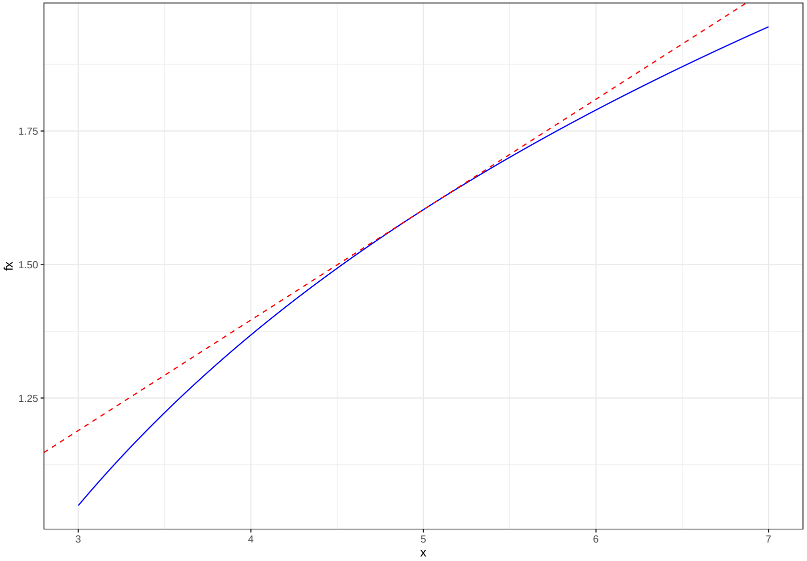
```r
fprime(5)
```

```
## [1] 0.2067379
```

We can see that $f(5) = 1.6027$ and $f'(5) = 0.2067379$

Copyright Miles Chen. For personal use only. Do not distribute.

Plot of $f(x)$ around $x = 5$



Copyright Miles Chen. For personal use only. Do not distribute.

Avoiding derivation

In the previous example, finding the derivative of $f(x)$ was relatively straightforward.

Sometimes finding the derivative of a function can be difficult.

In these cases, we can approximate the derivative numerically.

Copyright Miles Chen. For personal use only. Do not distribute.

Derivative definition

Let f be a Real valued function. The derivative of f at a is:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

We will use this definition and approximate it numerically using a suitably small value of h

Copyright Miles Chen. For personal use only. Do not distribute.

Approximate derivative

In R, we can implement this quite easily

```r
fprime_approx <- function(x){
  h <- 1e-6
  fx <- log(x) - exp(-x)
  fx_h <- log(x + h) - exp(-(x + h))
  (fx_h - fx) / h
}
print(fprime_approx(5), digits = 12) # the approximate value
```

```
## [1] 0.206737923669
```

```r
print(fprime(5), digits = 12) # the 'true' value of f'(5)
```

```
## [1] 0.206737946999
```

We see the approximation and the true value are very close in size

Copyright Miles Chen. For personal use only. Do not distribute.

Left or Right side derivative?

On the previous slide, I approximated the derivative by using a small positive h. If the derivative is continuous, we could also approximate the derivative using a small negative value of h.

```r
fprime_approx_neg <- function(x){
  h <- -1e-6 # h is negative
  fx <- log(x) - exp(-x)
  fx_h <- log(x + h) - exp(-(x + h))
  (fx_h - fx) / h
}
```

Copyright Miles Chen. For personal use only. Do not distribute.

Left or Right side derivative?

```r
print(fprime(5), digits = 12) # true value
```

```
## [1] 0.206737946999
```

```r
print(fprime_approx_neg(5), digits = 12) # left side approximation is too big
```

```
## [1] 0.206737970299
```

```r
print(fprime_approx(5), digits = 12) # right side approximation is too small
```

```
## [1] 0.206737923669
```

If there is curvature in the function, the left-side approximation and the right-side approximation of the derivative will differ slightly. One side will often be larger than the true value, and the other side will be smaller than the true value.

To correct this issue, when approximating the derivative numerically, it is recommended to look at both the right and left-side derivative approximations.

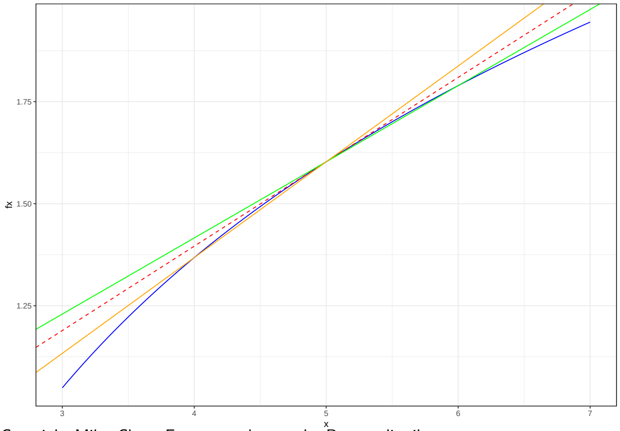
Copyright Miles Chen. For personal use only. Do not distribute.

A grossly exaggerated picture

The true slope of the tangent line at $x = 5$ is the red dashed-line.

The left-side approximation in orange is too steep. $(h = -1)$

The right-side approximation in green is not steep enough. $(h = 1)$



Copyright Miles Chen. For personal use only. Do not distribute.

Balancing the left and right side

To balance the left and right-side approximations, we can drawing a secant line from the coordinate $(x - h, f(x - h))$ to $(x + h, f(x + h))$.

```r
fprime_bal_approx <- function(x){
  h <- 1e-6
  fxplus <- log(x + h) - exp(-(x + h))
  fxminus <- log(x - h) - exp(-(x - h))
  (fxplus - fxminus) / (2 * h)
}
print(fprime_bal_approx(5), digits = 12) # balanced approximatin
```

```
## [1] 0.206737946984
```

```r
print(fprime(5), digits = 12) # true value
```

```
## [1] 0.206737946999
```

We see that the balanced approximation is even closer to the true value.

Copyright Miles Chen. For personal use only. Do not distribute.

Generalizing our derivative function

We can generalize the calculation of a derivative to work with any function.

```r
f_deriv <- function(ftn, x, h = 1e-6) {
  left <- x - h
  right <- x + h
  (ftn(right) - ftn(left)) / (2 * h)
}
```

The derivative function accepts an arbitrary function `ftn`, a location `x`, and value to use for h.

It is not necessary to analytically calculate the derivative. We simply estimate the derivative by calculating the value of the function at two locations and finding the slope.

Copyright Miles Chen. For personal use only. Do not distribute.

Derivative

```r
f <- function(x){
  log(x) - exp(-x)
}
print(f_deriv(f, 5), digits = 12) # our numeric derivative function
```

```
## [1] 0.206737946984
```

```r
f2 <- function(x) {
  x ^ 2
}
f_deriv(f2, 5) # the true derivative is 2 * x = 10, and it appears equal to it
```

```
## [1] 10
```

```r
print(f_deriv(f2, 5), digits = 12) # at digits = 12, we see that difference
```

```
## [1] 10.0000000014
```

Copyright Miles Chen. For personal use only. Do not distribute.

Care with floating point values

In the function we wrote, we can specify a different value for h. Be careful not to pick a value of h that is too small as we will run into the limits of the computer's floating point capabilities.

```
options(digits = 16)
f_deriv(f2, 5, h = 1e-5)
```

```
## [1] 9.999999999621423
```

```
f_deriv(f2, 5, h = 1e-6)
```

```
## [1] 10.00000000139778
```

```
f_deriv(f2, 5, h = 1e-7)
```

```
## [1] 10.00000002804313
```

```
f_deriv(f2, 5, h = 1e-8)
```

```
## [1] 9.99999993922529
```

Copyright Miles Chen. For personal use only. Do not distribute.

Floating Point Limitations

```r
f_deriv(f2, 5, h = 1e-10)
```

```
## [1] 10.00000082740371
```

```r
f_deriv(f2, 5, h = 1e-12)
```

```
## [1] 10.00088900582341
```

```r
f_deriv(f2, 5, h = 1e-14)
```

```
## [1] 9.769962616701378
```

```r
f_deriv(f2, 5, h = 1e-15)
```

```
## [1] 8.881784197001252
```

```r
f_deriv(f2, 5, h = 1e-16)
```

```
## [1] 0
```

Copyright Miles Chen. For personal use only. Do not distribute.

Floating Point Limitations

In our brief experiment, it seems that using a value around 10^{-6} is a good choice.

We see that when $h = 10^{-14}$ the accuracy drop dramatically, and the calculation returns 0 when $h = 10^{-16}$

Copyright Miles Chen. For personal use only. Do not distribute.

Section 3

Derivative-based Numeric Optimization

Copyright Miles Chen. For personal use only. Do not distribute.

Numeric Optimization

Numeric optimization methods are used to find a local minimum or local maximum of a function.

Many algorithms are written for finding a local min of $f(x)$. These methods can also be used to find a local max by applying the algorithm to $-f(x)$.

Copyright Miles Chen. For personal use only. Do not distribute.

Derivative-based Numeric Optimization

Derivative-based optimization can be used to find a local min or local max of a function $f(x)$. It is achieved by finding the roots of the derivative of $f'(x)$. The roots of the derivative will provide critical points that will frequently correspond to a local min or local max.

1. Find the derivative of $f(x)$
2. Apply any root finding algorithm to $f'(x)$ to find the roots.
3. Evaluate critical points.

Copyright Miles Chen. For personal use only. Do not distribute.

Common Method: Newton-Raphson for optimization

Find the roots of $f'(x)$ with the Newton-Raphson method.

1. Start at an arbitrary starting location x_0 that is expected to be "close enough" to the local min or local max.
2. Iteratively find the next value.

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

3. Stop when $|f'(x_n)| < \varepsilon$

Conditions:

- $f(x)$ needs to be twice differentiable
- $f''(x)$ cannot be 0 for any x_n that we can check. $f''(x)$ can be 0 as long as it is not in the sequence of locations x_n that get evaluated.

Copyright Miles Chen. For personal use only. Do not distribute.

Example

$f(x) = 0.25x^4 - 0.5x^2$

Analytic first and second derivative:

$f'(x) = x^3 - x$

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

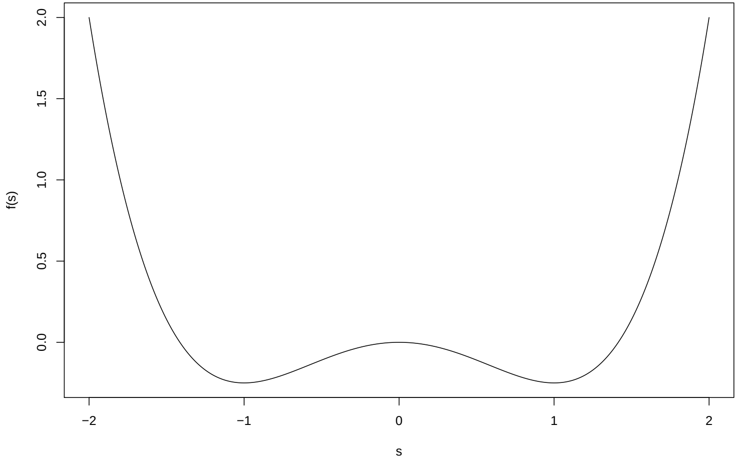
Copyright Miles Chen. For personal use only. Do not distribute.

Find the roots of $f'(x)$

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

Copyright Miles Chen. For personal use only. Do not distribute.

```r
f <- function(x){0.25 * x^4 - 0.5 * x^2}
s <- seq(-2,2, by = 0.01)
plot(s, f(s), type = "l")
```



Copyright Miles Chen. For personal use only. Do not distribute.

```r
fprime <- function(x){x^3 - x}
fdblprime <- function(x){3 * x^2 - 1}
xold <- 2.1
xnew <- xold - fprime(xold) / fdblprime(xold)
tol = 1e-6
while(abs(xnew-xold) > tol){
  xold <- xnew
  xnew <- xold - fprime(xold) / fdblprime(xold)
}
xnew
```

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

Copyright Miles Chen. For personal use only. Do not distribute.

```r
xold <- 0.9
xnew <- xold - fprime(xold) / fdblprime(xold)
tol = 1e-6
while(abs(xnew-xold) > tol){
  xold <- xnew
  xnew <- xold - fprime(xold) / fdblprime(xold)
}
xnew
```

```
## [1] 1.000000000000308
```

```r
xold <- 0.1
xnew <- xold - fprime(xold) / fdblprime(xold)
tol = 1e-6
while(abs(xnew-xold) > tol){
  xold <- xnew
  xnew <- xold - fprime(xold) / fdblprime(xold)
}
xnew
```

```
## [1] -9.926167350636332e-24
```

Copyright Miles Chen. For personal use only. Do not distribute.

```r
xold <- -10
xnew <- xold - fprime(xold) / fdblprime(xold)
tol = 1e-6
while(abs(xnew-xold) > tol){
  xold <- xnew
  xnew <- xold - fprime(xold) / fdblprime(xold)
}
xnew
```

```
## [1] -1
```

Copyright Miles Chen. For personal use only. Do not distribute.