Newton's Method

Boyie Chen

11/17/2019

Newton's Method

Concept

Given a function $f(x)$. Recall Taylor Expansion, we can approximate the function at a as the following:

$$f(x) = f(a)\frac{(x-a)^0}{0!} + f'(a)\frac{(x-a)^1}{1!} + \cdots + f^{(n)}(a)\frac{(x-a)^n}{n!} + \dots$$

Be careful that we need $f(x)$ to be smooth enough.

If we want to solve a function, i.e. to find x^* s.t. $f(x^*) = 0$, we can simply adopt the approximation above. Say we only use the first 2 terms to approximate $f(x^*)$:

$$0 = f(x^*) = f(a) + f'(a)(x^* - a)$$

Remember that the equality here is only for convenience.

After rearranging the above terms, we have:

$$af'(a) - f(a) = f'(a)x^*$$

And then we can find the x^* s.t. $f(x^*) = 0$ given a:

$$x^* = a - \frac{f(a)}{f'(a)}$$

Actually this equation (always remember the equality here is just for convenience) tells us the iterative relation between the optimal x^* and arbitrary starting point a. We can rewrite the above equation as a recursive version:

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

If $f(x)$ satisfies some good properties, then the sequence $\{x_n\}_{n=0}^{\infty}$ will converge. Thus we can find the solution x^* s.t. $f(x^*) = 0$. This method is called Newton's Method. It is widely used in many field of knowledge. The most relative concept in Machine Learning is "Gradient Descent" which is almost the same concept of Newton's Method.

Usage

1. First, guess a initial point x_0
2. Calculate x_1 by the recursive formula:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

3. Calculate x_2 by the same process, and on and on and on.
4. If the distance of x_{n-1} and x_n is small enough, then we can stop the iteration. The "small distance" is decided arbitrarily.

Requirements of Newton's Method:

- Smooth function
- Good inirial guess

Example 1

$$f(x) = x^3 + 2x^2 + 7$$

$$f'(x) = 3x^2 + 4x$$

We can define functions to calculate the value and the derivatives respectively.

```
#eg1. #f(x) = x^3+2*x^2-7
f = function(x){
   y = x^3+2*x^2-7; return(y)
}
f.prime = function(x){
  y = 3*x^2+4*x; return(y)
}
```

Given different input, the function returns different values.

```
x0 = 0
f(0); f(1); f(2); f(6)
```

```
## [1] -7
```

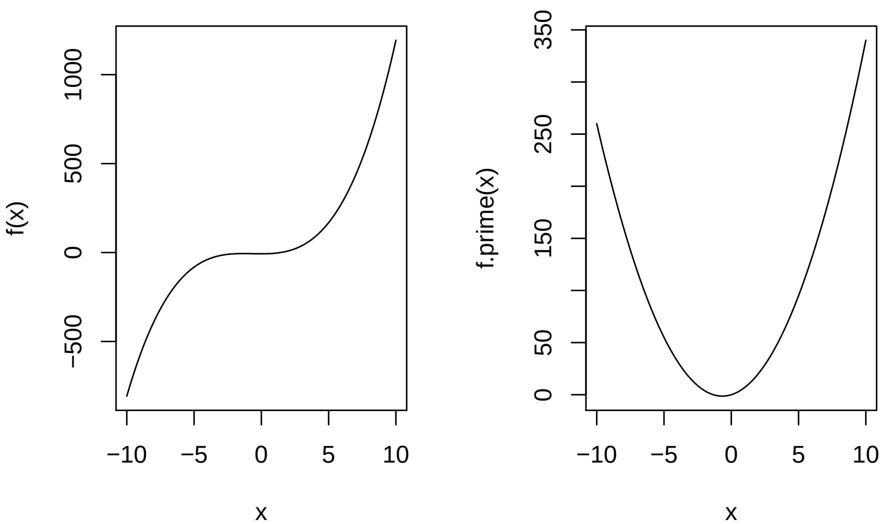
```
## [1] -4
```

```
## [1] 9
```

```
## [1] 281
```

Let's look at the graph of the function.

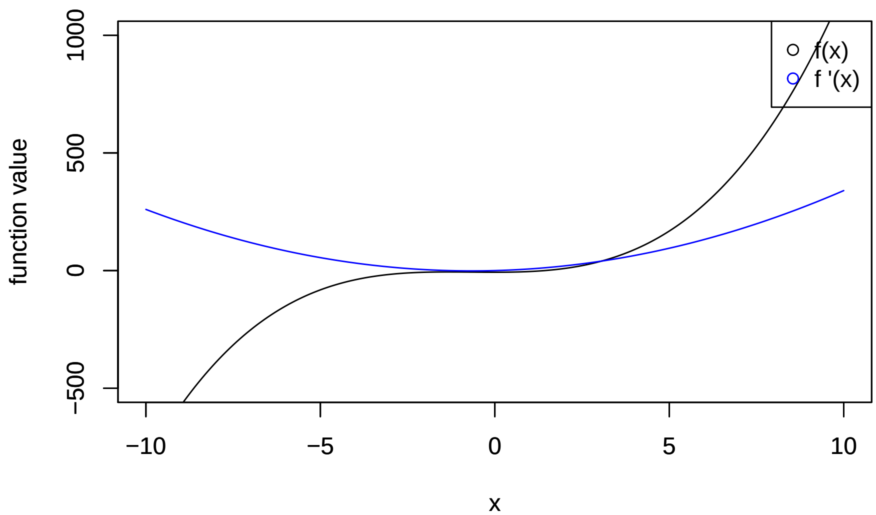
```
#have a glance
par(mfrow = c(1,2))
x = seq(-10, 10, by = 0.1)
plot(x, f(x), type = 'l')
plot(x, f.prime(x), type = 'l')
```



```r
graphics.off()
```

We can plot $f(x)$ and $f'(x)$ in the same graph.

```r
##overlapped view
plot(x, f(x), type = 'l', ylim = c(-500, 1000), ylab  = 'function value')
par(new = T)
plot(x, f.prime(x), type = 'l', ylim = c(-500, 1000), col = 'blue', ylab = 'function value')
legend("topright", # 表示在右上角
       pch = 1,     # pch 代表點的圖案
       col = c("black", "blue"), # col 代表顏色
       legend = c("f(x)", "f '(x)") # 顏色所對應的名稱
)
```




```r
graphics.off()
```

Iteration

With Newton's Method, we know the iteration is:

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

Let initial / starting point be $x_0 = 2$

```r
#initial / starting point x_0
x = 2
#recursive formula, first iteration
f=x^3+2*x^2-7
f.prime = 3*x^2+4*x
x = x - f/f.prime
```

Now we have x_1

```r
x
```

```
## [1] 1.55
```

Repeat the above process.

```r
#second iteration
f=x^3+2*x^2-7
f.prime = 3*x^2+4*x
x = x - f/f.prime
x #x_2
```

```
## [1] 1.435969
```

And we get x_2. Keep iterating.

```r
#repeat the following 3 lines and get the root
f=x^3+2*x^2-7
f.prime = 3*x^2+4*x
x = x - f/f.prime
x #x_3
```

```
## [1] 1.428845
```

Using loop with `while()`

```r
#Try to write the above repeating things in a loop
x = 2
tolerance = 10^(-6)
while(abs(f) > tolerance){
  x = x - f/f.prime
  f = x^3+2*x^2-7
  f.prime = 3*x^2+4*x
}
x
```

```
## [1] 1.428818
```

Example 2

$$f(x) = x^m$$

find x^* s.t. $f(x^*) = c$, i.e. find x^* s.t. $f(x^*) - c = 0$

This question is equivalent to find the m^{th} root of c, $c^{\frac{1}{m}}$. （等同於尋找 c 的 m 次方根, $c^{\frac{1}{m}}$）

```r
#Let's write a function to solve x^m = c
#i.e. find one of the root of x^m - c = 0
#i.e. c^(1/m)
findRoot = function(m = numeric('power'), c = numeric()){
  x = 1
  tolerance = 10^(-6)
  f = x^m - c
  f.prime = m*x^(m-1)

  while(abs(f) > tolerance){
    x = x - f/f.prime
    f = x^m - c
    f.prime = m*x^(m-1)
  }
  return(x)
}
```

Let's try if this function works well.

```
findRoot(5, 10) # 誰的五次方是 10 #only find the real root
```

```
## [1] 1.584893
```

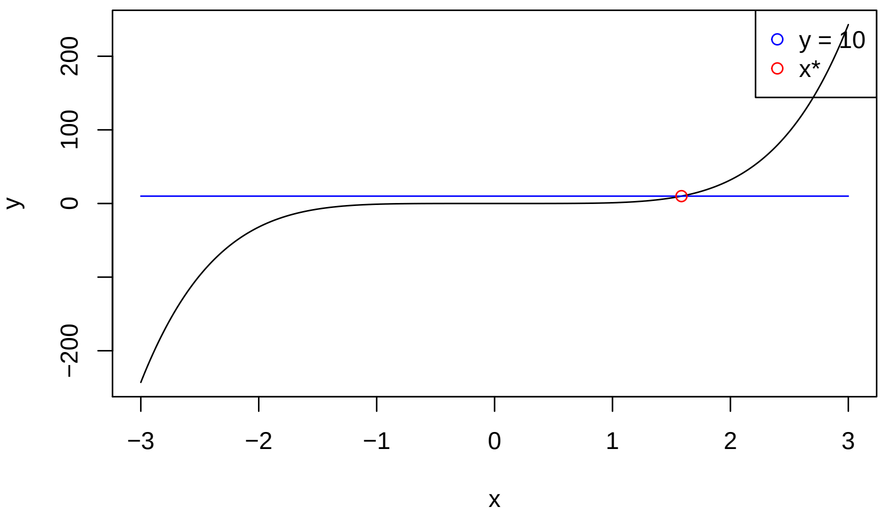
```
1.584893^5 # 驗算
```

```
## [1] 9.999994
```

```
10^(1/5) # 小型工程計算機就是用類似的方法寫的
```

```
## [1] 1.584893
```

```
#visualization
x = seq(-3, 3, by = 0.01)
y = x^5 #f(x)
plot(x, y, type = 'l')
lines(x, rep(10, length(x)), col = 'blue') #f(x) = 10
points(findRoot(5, 10), 10, col = 'red')
legend("topright", # 表示在右上角
       pch = 1,     # pch 代表點的圖案
       col = c("blue", "red"), # col 代表顏色
       legend = c("y = 10", "x*") # 顏色所對應的名稱
)
```



When Newton's Method Fails

If some criteria are not satisfied:

- The function is not smooth enough, or it is not differentiable
- The initial guess is not good enough

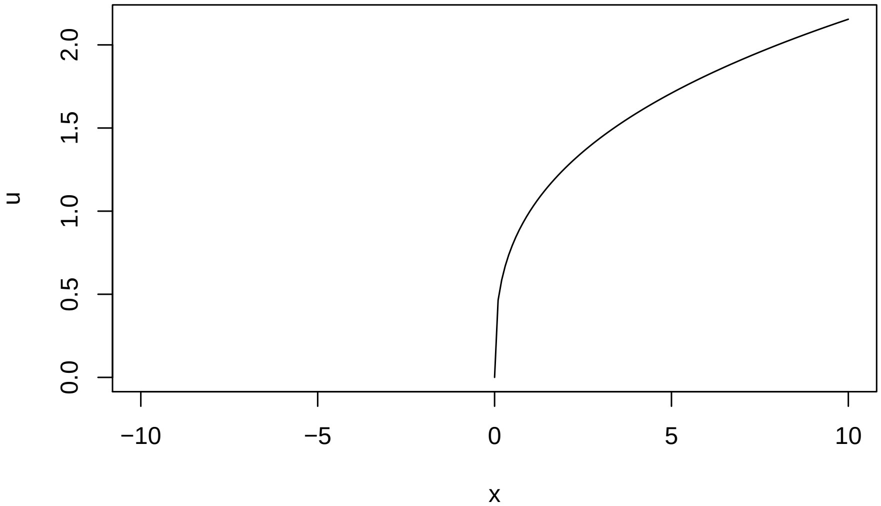
Example : Frequent Used Utility Function

$$u(x) = x^{\frac{1}{3}}$$

$$u'(x) = \frac{1}{3}x^{\frac{-2}{3}}$$

Note that $u'(x)$ is undefined when $x = 0$

```r
#when starting point is not good enough #f(x) = x^(1/3)
x = seq(-10, 10, by = 0.1)
u = x^(1/3)
plot(x, u, type = 'l')
```



I'll use `error = T` in RMarkdown to show that the following code will return error after second iteration.

```r
#find root #will fail
x = 1
u = x^(1/3)
u.prime = (1/3)*x^(-2/3)
tolerance = 10^(-6)
while(abs(u)>tolerance){
  x = x - u/u.prime
  u = x^(1/3)
  u.prime = (1/3)*x^(-2/3)
}
```

```
## Error in while (abs(u) > tolerance) {: 需要 TRUE/FALSE 值的地方有缺值
```

```r
x #the iteration fails, so this x is not the optimal
```

```
## [1] -2
```

Conclusion

The concept of Newton's Method is finding the solution with the guiding of first order derivatives. Thus the objective function must first satisfy some properties. You should always check whther the objective function you want to maximize satisfies these criteria or not.

Alternative View & Details of Numeric Mximization/Minimization (Optional)

We can definitely write a function that do the `while` loop above. The input of the function is a function, its derivative and an initial guess.

```r
#another version that you can decide your starting point
root = function(f, f.prime, guess) {
  tolerance = 10^(-6) #tol
```

```
  x = guess
  while (abs(f(x)) > tolerance) {
    x = x - f(x)/f.prime(x)
  }
  return(x)
}
```

Given the same function in example (1): $f(x) = x^3 + 2x^2 + 7$ and its derivative $f'(x) = 3x^2 + 4x$

```
f = function(x) {x^3 + 2*x^2 - 7}
f.prime = function(x) {3*x^2 + 4*x}
```

With a initial guess $x_0 = 10$, we can find the root x^* s.t. $f(x^*) = 0$

```
root(f, f.prime, 10)
```

```
## [1] 1.428818
```

Different Tolerances Give Different Decimal Digits

I define `tolerance = 10^(-6)` in the above function. Sometimes, this extent of tolerance is not precise enough.

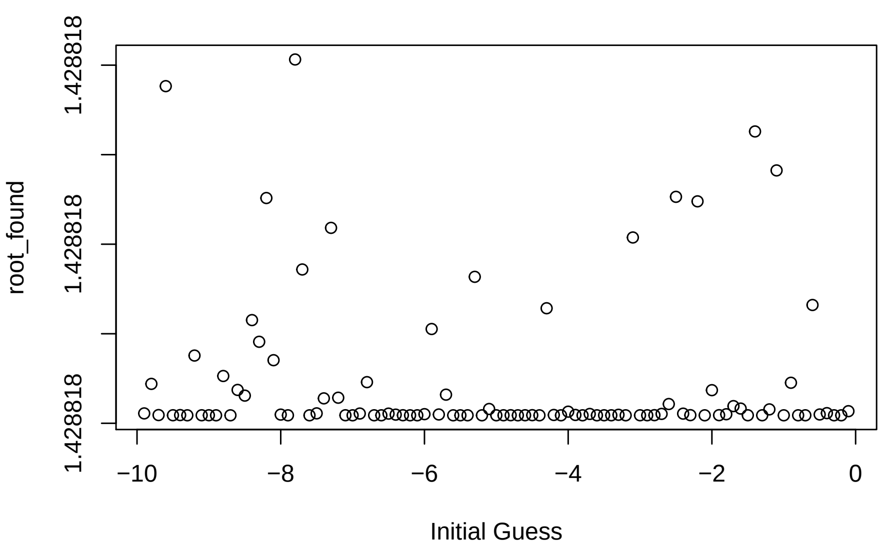
```
x = rep(0, 99)
root_found = c()
for(i in 1:99){
  x[i] = -10+ 0.1*i
  root_found = cbind(root_found, root(f, f.prime, x[i]))
}
length(x)
```

```
## [1] 99
```

```
length(root_found)
```

```
## [1] 99
```

```
plot(x[1:99], root_found, xlab = 'Initial Guess') #redo this line after changing tol
```



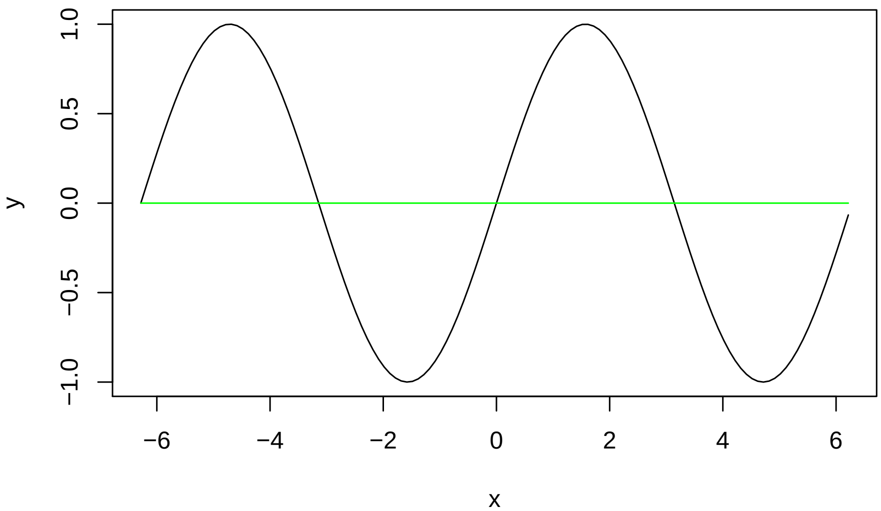
We can see that even though all x^* we found are close to 1.428818, many of them deviate from this value given different initial guess.

Other Situations that Newton's Method Fails

Given $f(x) = sin(x)$, and $f'(x) = cos(x)$ The function above is periodic on $\mathcal{R}$ and it has periodicity of 2π.

If we want to find the roots of $f(x) = 0$, we will find a bunch of x s.t. $sin(x) = 0$.

```
#Sin function
x = seq(-2*pi, 2*pi, by = 0.1)
y = sin(x)
plot(x, y, type = 'l')
lines(x, rep(0, length(x)), col = 'green')
```



What we will find depends on our initial guess.

```
#the numerical approach depends on initial guess
root(sin, cos, 3) #derivative of sin is cos #start from 3 => find pi
```

```
## [1] 3.141593
```

```
root(sin, cos, -3) #start from -3 => find -pi
```

```
## [1] -3.141593
```

Also, if we unfortunately choose the initial point x_0 at $\frac{\pi}{2}$ or its multiples, we will have an infinite value derivative. In this kind of situation, we cannot find the root because we can not even find x_1.

```
#The following command is a wrong example
#root(sin, cos, pi/2) #caution: infty loop
```

Even though we are not choosing x_0 at $\frac{\pi}{2}$ or its multiples, we may have a x_1 far away from x_0. If the function does not satisfy some properties globally, we may not find the solution.

```
root(sin, cos, pi/2+0.001) #jump really far away
```

```
## [1] 1002.168
```

Frequent Used Non-Linear Minimiaztion Function

Sometimes, we're not only interested in x^* s.t. $f(x^*) = 0$, but also those $\hat{x}$ s.t. $f(\hat{x})$ is the global/local maximum/minimum or saddle point.

```
#introduce `nlm` : minimize non-linear model
nlm(sin, pi, hessian = T) #give pi => goes to the right and find 1.5*pi
```

```
## $minimum
## [1] -1
##
## $estimate
## [1] 4.712387
##
## $gradient
## [1] 0
##
## $hessian
##           [,1]
## [1,] 0.9999999
##
## $code
## [1] 1
##
## $iterations
## [1] 4
```

```r
4.712387/pi
```

```
## [1] 1.499999
```

The above function `nlm()` needs an objective function & an initial guess(parameters) as inputs. It will adjust the parameters and find the minimized value of the objective function.

We give `nlm()` a function $sin(x)$ and initial guess $x_0 = \pi$. It gives us back 4.712387 which is nearly 1.5π ,or 3 times $\frac{\pi}{2}$. The $sin(x)$ encounter a local maximum/minimum every $\frac{\pi}{2}$.

We will use this `nlm()` function in finding the Maximized Likelihood Estimator.