

2019-09-18

computational Statistics

HW#1

192STG11 우나영

computational Statistics

HW#1

1. Problem

최적화(Optimization)는 통계 추론에서 주요한 문제 중 하나이다. 최적화는 통계 추론의 핵심 중의 하나인 최대 우도(Maximum likelihood)를 구하는 문제 뿐만 아니라 베이지안의 결정 이론에서 위험(Bayes risk)을 줄이는 문제, 가장 높은 사후분포의 확률 구간을 구하는 문제, 그리고 비선형 최소 제곱법을 푸는 문제 등 다양한 통계학의 분야에서 문제 해결의 핵심적인 열쇠가 된다. 최적화 문제는 실수치의(real-valued) 함수 g 를 최적화하는 p 차원의 x vector 의 값을 구하는 문제이다. 예를 들어, 최대 우도 추정량을 계산할 때 log-likelihood 함수 ℓ 은 실수치의 함수 g 에 대응되고 모수 θ vector 는 p 차원의 x vector 에 대응된다. 이때 $\hat{\theta}$ 이 최대 우도 추정량(Maximum likelihood estimator)라면 $\hat{\theta}$ 가 log-likelihood 함수 ℓ 을 최적화하는 값이 된다. 따라서 $\hat{\theta}$ 은 log-likelihood 함수 ℓ 의 1차 미분 함수인 score function ℓ' 의 해가 된다. 즉 최적화 문제는 결국 함수의 근을 찾는 문제로 이어진다. 대부분의 함수들은 닫힌 해(closed form solution)를 갖지 않기 때문에 반복적인 근사(iterative approximation)를 통해 해를 찾는 알고리즘을 사용할 것이다.

본 과제에서 R 을 이용하여 해를 찾는 세가지 대표적인 알고리즘으로 Bisection, Newton-Raphson 그리고 Secant 를 구현할 것이다. 세 알고리즘은 공통적으로 초기값(starting value)에서 출발하여 기존의 x 값(x_t)에서 새로운 x 값(x_{t+1})으로 갱신(update)하면서 해 값에 근사한다. 이때 초기값은 그래프를 통해 시각적으로 어림 잡아 구하였다. 또한 stopping rule 은 $|x_{t+1} - x_t| < \epsilon$ where $\epsilon = 10^{-10}$ 으로 absolute convergence criterion 을 사용하였다. 세 알고리즘의 차이점은 새로운 x 값을 갱신(update)하는 방식이다. Bisection 은 근이 존재하는 폐구간을 이분한 후, 이 중 근이 존재하는 하위 구간을 선택하는 것을 반복하여 해의 값에 근사하는 방법이다. 원리는 만약 함수 g' 가 구간 $[a, b]$ 에서 연속이고 $g'(a)g'(b) \leq 0$ 이라면 중간값의 정리(intermediate value theorem)에 따라 구간 안에 최소한 하나의 해를 가진다. Bisection 의 초기값은 해가 존재할 것으로 예상되는 구간 $[a, b]$ 이다. 구간으로부터 해의 초기값을 $x_t = \frac{(b-a)}{2}$ 으로 잡고 아래의 updating equation 에 따라 구간과 x 의 값을 반복적으로 갱신시켜 해 값에 근사 한다.

$$[a_{t+1}, b_{t+1}] = \begin{cases} [a_t, x_t] & \text{if } g'(a_t)g'(x_t) \leq 0 \\ [x_t, b_t] & \text{if } g'(x_t)g'(b_t) \leq 0 \end{cases}, x_{t+1} = \frac{(a_{t+1} + b_{t+1})}{2}$$

Newton-Raphson 은 g' 이 연속 미분 함수이며 $g''(x^*) \neq 0$ 일 때, 해의 초기값 x_t 에서 접선을 그린 후 접선이 x 축과 만나는 지점으로 x 를 이동시켜 점진적으로 해의 값으로 근사하는 방법이다. 이때 $g''(x^*)$ 은 Taylor series expansion 에 근사 시켜 다음과 같은 새로운 x 값의 updating rule 을 따른다.

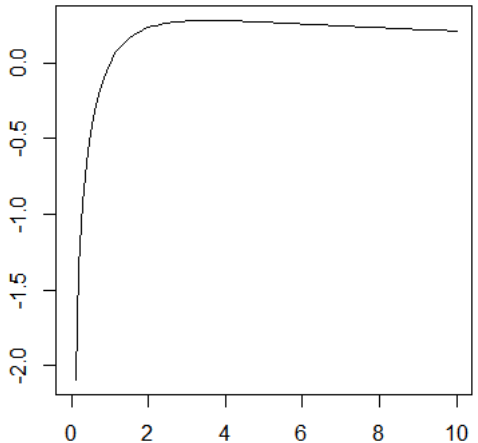
$$0 = g'(x^*) \approx g'(x_t) + (x^* - x_t)g''(x_t)$$

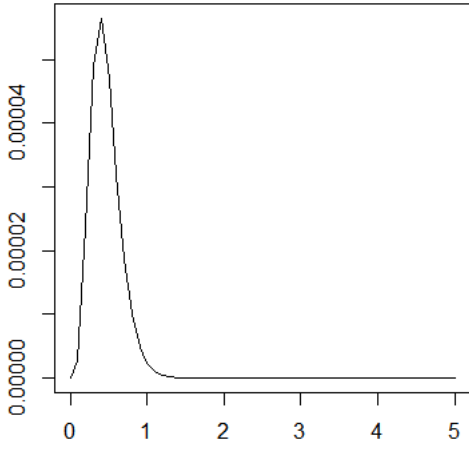
$$x_{t+1} = x_t - \frac{g'(x_t)}{g''(x_t)}$$

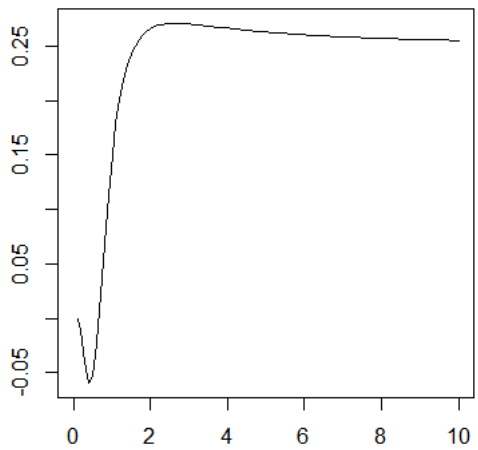
Secant 는 Newton-Raphson 의 접선 대신 할선을 이용하여 해를 찾는 방법이다. 해의 초기값은 할선의 두 점인 x_{t-1} 과 x_t 이다. 원리는 Newton-Raphson 과 같이 초기값 x_{t-1} 과 x_t 에서 할선을 그린 후 할선이 x 축과 만나는 지점으로 x 를 이동시켜 점진적으로 해의 값으로 근사시키는 방법이다. 아래와 같은 새로운 x 값의 updating rule 을 따른다.

$$x_{t+1} = x_t - g'(x_t) \frac{x_t - x_{t-1}}{g'(x_t) - g'(x_{t-1})}$$

2. Result

| # example 1 # | | | |
|------------------------------|---|-------------------|-------------------|
| $g(x) = \frac{\log(x)}{1+x}$ | <p style="text-align: center;">plot of $\log(x)/(1+x)$</p>  | | |
| | Bisection | Newton | Secant |
| initial value | (1, 5) | 2 | (1, 5) |
| x^* | 3.59112147666747 | 3.5911214766659 | 3.59112147665764 |
| $g(x^*)$ | 0.278464542761074 | 0.278464542761074 | 0.278464542761074 |
| The number of iteration | 35 | 8 | 15 |
| Time(sec) | 0.015 | 0.0056 | 0.0092 |

| # example 2 # | | | | |
|---|--|-----------------------|-----------------------|-----------------------|
| $f(\lambda) = \lambda^5 e^{-13\lambda}$ | <p>plot of $(x^5)*\exp(-13*x)$</p>  | | | |
| | Bisection | Newton | Secant | |
| | initial value | (0.1, 1) | 0.3 | (0.1, 0.5) |
| | λ^* | 0.384615384571953 | 0.384615384615311 | 0.384615384615467 |
| | $f(\lambda^*)$ | 0.0000567101571323512 | 0.0000567101571323512 | 0.0000567101571323512 |
| | The number of iteration | 33 | 5 | 8 |
| | Time(sec) | 0.0119 | 0.0028 | 0.0047 |

| # example 3 # | | | |
|---|--|------------------|------------------|
| $h(x) = \frac{\log(x) + x^2}{4x^2 + e^{\frac{1}{x}}}$ | <p>plot of $(\log(x)+x^2)/(4*x^2+\exp(1/x))$</p>  | | |
| | Bisection | Newton | Secant |
| initial value | (1, 5) | 2 | (1, 3) |
| x^* | 2.63865702407202 | 2.63865702403882 | 2.63865702395344 |

| | | | |
|-------------------------|-------------------|-------------------|-------------------|
| $h(x^*)$ | 0.270643264403539 | 0.270643264403539 | 0.270643264403539 |
| The number of iteration | 35 | 7 | 11 |
| Time(sec) | 0.0104 | 0.0031 | 0.0063 |

3. Discussion

| | 장점 | 단점 |
|-----------|---|--|
| Bisection | <ul style="list-style-type: none"> ✓ 1 개의 근을 무조건 구할 수 있다. ✓ 안정적이다(stable) ✓ 2 차 미분계수 값이 필요하지 않다. | <ul style="list-style-type: none"> ✓ 수렴속도가 가장 느리다 ✓ 초기치가 두개 |
| Newton | <ul style="list-style-type: none"> ✓ 수렴속도가 가장 빠르다. ✓ 초기치 값이 하나만 있으면 된다. | <ul style="list-style-type: none"> ✓ 2 차 미분계수 값이 필요함 ✓ 초기치에 민감하다(초기치 값에 따라 발산할 수 있다) |
| Secant | <ul style="list-style-type: none"> ✓ Newton-Raphson 방법과 달리 2 차 미분계수 값이 필요하지 않다 ✓ Newton 보다 느리지만 안정적(stable)하고 Bisection 보다 빠르다. | <ul style="list-style-type: none"> ✓ 초기치에 민감하다 ✓ 초기치가 두개 |

Bisection 은 초기값이 구간이기 때문에 두 개이고 Secant 는 할선을 그려줄 초기값 두 개가 필요하다. 반면에 Newton-Raphson 은 접선을 그려줄 초기값 하나만 필요하다. 수렴속도는 Newton-Raphson, Secant, 그리고 Bisection 순으로 빠르다. 알고리즘이 안정적인 것은 Bisection, Secant, 그리고 Newton-Raphson 순이다. 알고리즘이 안정적이라는 것은 초기값에 민감하지 않은 것을 말한다. Bisection 은 수렴속도가 느리고 초기치가 두 개가 필요하지만 2 차 미분계수가 필요하지 않아 계산이 간단하고 구간 안의 최소 1 개의 근을 무조건 구할 수 있어 근의 존재 자체가 중요한 목적일 경우 사용하기에 적합하다. Newton-Raphson 은 접선으로 해에 근사하기 때문에 굉장히 수렴속도가 빠르다. 그러나 접선을 이용하기 때문에 초기값에 따라 발산하거나 수렴하지 못하는 경우가 발생한다. 또한 2 차 미분계수가 필요하기 때문에 계산의 복잡성이 따른다. Secant 는 Newton-Raphson

4. Appendix

```
rm(list=ls())
library(numDeriv)
library(tidyverse)
options(scipen=999)

### Bisection function
bisec<-function(f, a, b){

  maxiter <- 1000
  threshold <- 10^(-10)
```

```
err <- 1
niter <- 0
x0 <- ( a + b ) / 2

while ( niter <= maxiter && err >= threshold){
  #update interval
  if ( (genD(func = f, x = a)$D[1]) * (genD(func = f , x = x0)$D[1]) <=0)      {b <- x0}
  else {a <- x0}

  #update x
  oldx0 <- x0
  x0 <- ( a + b )/2

  #update error and niter
  err<-abs( oldx0 - x0 )
  niter <- niter + 1
}
print(paste(paste0("Number of itertation is :",niter),paste0("x* is :",x0),paste0("f(x*)
is : ",f(x0)),sep="      "))
}

### Newton's function
newt <- function( f, x0 ){

  maxiter <- 1000
  threshold <- 10^(-10)
  err <- 1
  niter <- 0

  while ( niter <= maxiter && err >= threshold){

    #update x
    oldx0 <- x0
    x0 <- oldx0 - ( genD(f, oldx0)$D[1] / hessian(f, oldx0)[1] )

    #update error and niter
    err<-abs( oldx0 - x0 )
    niter <- niter + 1

  }
  print(paste(paste0("Number of itertation is :",niter),paste0("x* is :",x0),paste0("f(x*)
is : ",f(x0)),sep="      "))
}

### secant function
sec <- function( f, x0, x1 ){

  maxiter <- 1000
  threshold <- 10^(-10)
  err <- 1
  niter <- 0
  x2 <- 0

  while ( niter <= maxiter && err >= threshold){

    # calculate x2
    x2 <- x1 - genD(f, x1)$D[1] / ((genD(f, x1)$D[1] - genD(f, x0)$D[1])/(x1 - x0))
```

```
# update x0 and x1
x0 <- x1
x1 <- x2

#update error and niter
err <- abs( x1 - x0 )
niter <- niter + 1

}
print(paste(paste0("Number of itertation is :",niter),paste0("x* is :",x1),paste0("f(x*)
is : ",f(x1)),sep="
"))
}

### example 1
f1 <- function(x){
  log(x)/(1+x)
}
xx <- seq(0,10,by=0.1)
plot(xx,f1(xx),type='l',xlab=" ", ylab=" ", main="plot of log(x)/(1+x)")

system.time(for (i in 1:100)bisec(f1,1,5))
system.time(for (i in 1:100)newt(f1, 2))
system.time(for (i in 1:100)sec(f1, 1, 5))

### example 2
f2 <- function(x){
  (x^5)*exp(-13*x)
}
xx2 <- seq(0,5,by=0.1)
plot(xx2,f2(xx2),type='l',xlab=" ", ylab=" ", main="plot of (x^5)*exp(-13*x)")

system.time(for (i in 1:100)bisec(f2,0.1,1))
system.time(for (i in 1:100)newt(f2, 0.3))
system.time(for (i in 1:100)sec(f2, 0.1, 0.5))

### example 3
f3 <- function(x){
  ( log(x) + x^2 )/(4 * x^2 + exp(1/x) )
}
xx3 <- seq(0,10,by=0.1)
plot(xx3, f3(xx3), type='l', xlab=" ", ylab=" ", main="plot of (log(x)+x^2)/(4*x^2+exp(1/x))")

system.time(for (i in 1:100)bisec(f3, 1, 5))
system.time(for (i in 1:100)newt(f3, 2))
system.time(for (i in 1:100)sec(f3, 1, 3))
```