BSC02

Pan Gaojie

2023/10/29

目录

Newton-Raphson Method **1**

Pre-work 1

Code a function to accomplish Newton method for root finding. 2

Practical application **3**

Root finding 3

Optimization 6

Appendix **7**

Newton-Raphson Method

Pre-work

```
#The function can assign values to multivariate expressions.
#giving expression(expr) and point(value)
exprfunc<-function(expr,values){
  params<-all.vars(expr)
  #assign values to params
  for(i in c(1:length(params))){
    assign(params[i],values[i],envir = .GlobalEnv)
  }
}
```

```r
#Multiple dimension deriv() calculator with assignment to parameters
Myderiv<-function(expr, values, d=1){
  #d=1 represents first derivative etc.
  n<-length(values)
  exprfunc(expr, values)
  de<-deriv(expr, all.vars(expr),hessian = TRUE)
  if(d==0) return(eval(expr))
  if(d==1) return(matrix(attr(eval(de),"gradient"),nr=n,nc=1))
  if(d==2) return(matrix(attr(eval(de),"hessian"),nr=n,nc=n))
}
```

Code a function to accomplish Newton method for root finding.

```r
newton.rf<-function(expr, init, optim = FALSE){
  #expr represents the expression that root-finding interests in.
  #init represents the initial point.
  #optim represents if this function used to optimize the expression or not.
  Error_accepted<-10^-9
  epsilon<-1
  #these value sets stop condition abs(epsilon)<E_accepted
  #epsilon represents |dfx|
  if(optim == FALSE) option=0
  if(optim == TRUE) option=1
  #option controls d in Myderiv
  root<-init
  while(abs(epsilon)>=Error_accepted){
    fx<-Myderiv(expr,root,d=option)
    dfx<-Myderiv(expr,root,d=option+1)
    root<-root-solve(dfx)%*%fx
    epsilon<-norm(as.matrix(fx))
  }
  return(root)
}
```

Practical application

Root finding

1.Consider $f(x) = x^2 - 5$

```
expr1<-expression(x^2-5)
newton.rf(expr1,1)
```

```
##          [,1]
## [1,] 2.236068
```

```
newton.rf(expr1,-1)
```

```
##           [,1]
## [1,] -2.236068
```

2.Consider $f(x) = |x|^{1/2}$

```
expr2 <- expression(sqrt(abs(x)))
#newton.rf(expr2,0.25)
#Error in deriv.default(expr, all.vars(expr), hessian = TRUE) :
  # 微分表里没有这个函数'abs'
```

Due to the lack of differentiability at the origin, deriv() is unusable. Code a function exclusively for$f(x) = |x|^{1/2}$

```
f <- function(x) sqrt(abs(x))
f_prime <- function(x) {
  if (x > 0) {
    return(1 / (2 * sqrt(x)))
  } else if (x < 0) {
    return(-1 / (2 * sqrt(-x)))
  } else {
    return(NULL)
  }
}

newton_sqrt_abs <- function(initial_guess, tol = 10^-9, max_iter = 10) {
```

```
  x <- initial_guess
  for (i in 1:max_iter) {
    f_x <- f(x)
    f_prime_x <- f_prime(x)
    x_new <- x - f_x / f_prime_x
    if (abs(x_new - x) < tol) {
      return(x_new)
    }
    x <- x_new
    print(x)
  }
  return(NULL)
  # If the maximum number of iterations is reached
  #without convergence, return NULL.
}

result <- newton_sqrt_abs(initial_guess = 0.25)
```

```
## [1] -0.25
## [1] 0.25
## [1] -0.25
## [1] 0.25
## [1] -0.25
## [1] 0.25
## [1] -0.25
## [1] 0.25
## [1] -0.25
## [1] 0.25
```

```
print(result)
```

```
## NULL
```

We set that if the maximum number (1000) of iterations is reached without convergence, return NULL. And the function returns NULL in the end, which means the iteration is failed.

3.Consider $f(x) = xe^{-x^2} - 0.4(e^x + 1)^{-1} - 0.2$

```
expr3<-expression(x*exp(-x^2)-0.4*(exp(x)+1)^(-1)-0.2)
newton.rf(expr3,0.5)
```

```
##             [,1]
## [1,] 0.4303877
```

```
newton.rf(expr3,0.6)
```

```
##             [,1]
## [1,] 0.4303877
```

The starting points are set to 0.5 and 0.6 respectively, and the results are the same.

4.Consider $f(x) = x^3 - 2x^2 - 11x + 12$

```
expr4<-expression(x^3-2*x^2-11*x+12)
newton.rf(expr4,2.35287527)
```

```
##      [,1]
## [1,]    4
```

```
newton.rf(expr4,2.352884172)
```

```
##      [,1]
## [1,]   -3
```

The starting points are set to 0.5 and 0.6 respectively, and the results varies. And both results are valid.

5.Consider $f(x) = 2x^3 + 3x^2 + 5$

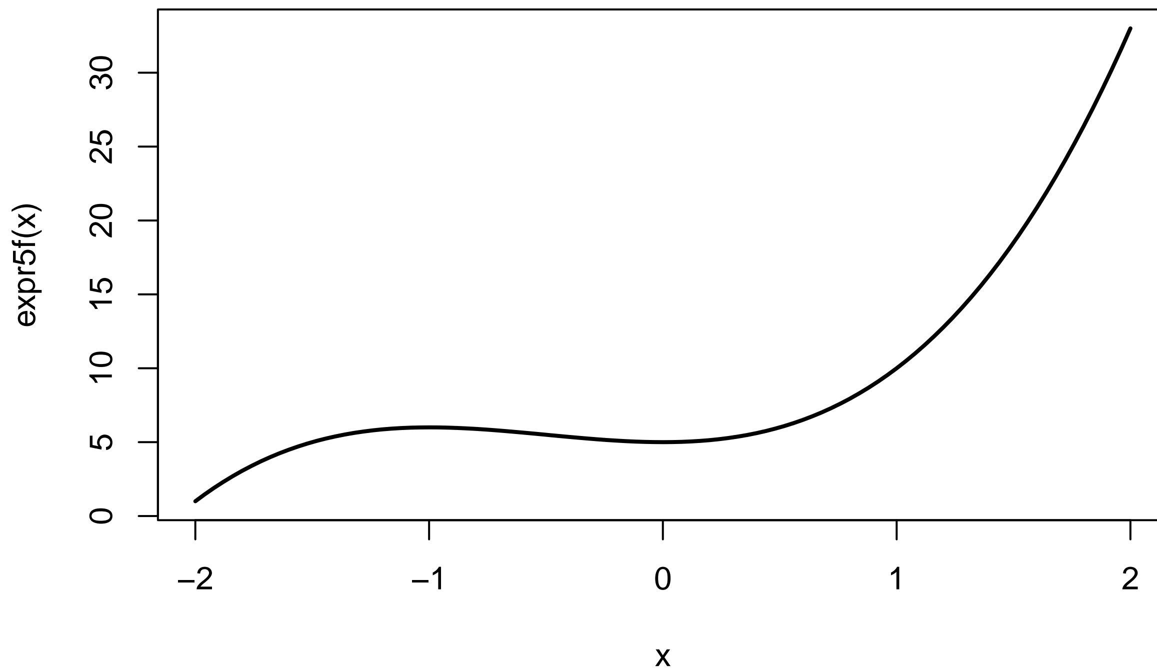
```
expr5<-expression(2*x^3+3*x^2+5)
newton.rf(expr5,0.5)
```

```
##             [,1]
## [1,] -2.078617
```

```
#newton.rf(expr5,0)
#Error in solve.default(dfx) :
#Lapack routine dgesv: system is exactly singular: U[1,1] = 0
```

Plot the expression and see what happening here.

```
expr5f <- function(x) 2*x^3 + 3*x^2 + 5
curve(expr = expr5f, from = -2, to = 2, lwd = 2, xlab = "x")
```



Evidence from the graph indicates the derivative at point 0 is 0, try to confirm that numerically.

```
Myderiv(expr5,0)
```

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

As the derivative is 0, it's clear why the function doesn't work.

Optimization

1.Consider $f(x) = -x^4$

```
expr6<-expression(-x^4)
newton.rf(expr6,1,optim = TRUE)
```

```
##                [,1]
## [1,] 0.0003007287
```

There's a deviation between result and 0. Use R's standard function nlm()

```
nlm(function(x) x^4,x)$estimate
```

```
## [1] 0.000451093
```

Notwithstanding usage of nlm(), there's still a deviation.

2.Consider $f(x) = x^2 - xy + y^2 + e^y$

```
expr7<-expression(x^2-x*y+y^2+exp(y))
newton.rf(expr7,c(0,0),optim = TRUE)
```

```
##              [,1]
## [1,] -0.2162814
## [2,] -0.4325628
```

Appendix

test the usage of deriv() function.

```
f<-expression(x+y)
f1<-deriv(f, c("x","y"), function.arg=TRUE, hessian = TRUE)
(f1)
```

```
## function (x, y)
## {
##     .value <- x + y
##     .grad <- array(0, c(length(.value), 2L), list(NULL, c("x",
##         "y")))
##     .hessian <- array(0, c(length(.value), 2L, 2L), list(NULL,
##         c("x", "y"), c("x", "y")))
##     .grad[, "x"] <- 1
##     .grad[, "y"] <- 1
##     attr(.value, "gradient") <- .grad
##     attr(.value, "hessian") <- .hessian
##     .value
## }
```

test eval() function.

```
exprfunc<-function(x) eval(expr)
expr<-expression(x^2)
exprfunc(1)
```

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

test eval() for multi-dimension cases

```
expr<-expression(x+y-z)
values<-c(1,2,4)
exprfunc<-function(expr,values){
  params<-all.vars(expr)
  #assign values to params
  for(i in c(1:length(params))){
    assign(params[i],values[i],envir = .GlobalEnv)
  }
}
exprfunc(expr,values)
eval(expr)
```

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

test the assignment to parameters in deriv()

```
f<-expression(x+y)
exprfunc(f,c(1,2))
f3<-deriv(f, all.vars(f),hessian = TRUE)
(grad<-matrix(attr(eval(f3),"gradient"),nr=2,nc=1))
```

```
##      [,1]
## [1,]    1
## [2,]    1
```

```
(hessian<-matrix(attr(eval(f3),"hessian"),nr=2,nc=2))
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```