21377174-屠心怡-第二次作业

21377174 屠心怡

2023-10-27

目录

1 Newton-Raphson code

根据牛顿法的基本原理，遵循迭代原则 $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ 并在给定的 tolerance level 下编写适用于一元和多元函数计算的方法。

```r
newton_method <- function(f, x0, tol=1e-6, max_iter=1000) {
  # 判断是否为多元函数
  is_multivariate <- is.matrix(x0) || length(x0) > 1

  # 如果是一元函数
  if (!is_multivariate) {
    for (i in 1:max_iter) {
      # 计算导数
      df_expr <- D(body(f), "x")
      df_func <- function(x) eval(df_expr)
      df <- df_func(x0)

      # 更新 x0
      x0 <- x0 - f(x0) / df

      # 检查收敛性
      if (abs(f(x0)) < tol) {
        return(x0)
      }
    }
  } else { # 如果是多元函数
    n <- length(x0)# 判断维数，n 元，n 个未知数
    for (i in 1:max_iter) {
      # 计算梯度
      gradient <- numeric(n)
      for (j in 1:n) {
        var_name <- paste0("x", j)# 命名变量为 xj
        df_expr <- D(as.expression(body(f)), var_name)
        df_func <- function(x1, x2, x3, x4, x5, ...) {
          eval(df_expr)
        }
        gradient[j] <- do.call(df_func, as.list(x0))
        #as,list(x0) 把 x0 转化为列表，再调用 do.call 把求微分函数作用于列表 X0
      }

      # 计算 Hessian 矩阵
```

```
      hessian <- matrix(0, nrow=n, ncol=n)
      for (j in 1:n) {
        for (k in 1:n) {
          var_name_j <- paste0("x", j)# 变量命名 xj
          var_name_k <- paste0("x", k)# 变量命名 xk
          d2f_expr <- D(D(body(f), var_name_j), var_name_k)# 先对 xj 求导再对 xk 求导
          d2f_func <- function(x1, x2, x3, x4, x5, ...) {
            eval(d2f_expr)
          }
          hessian[j, k] <- do.call(d2f_func, as.list(x0))
        }
      }

      # 更新 x0
      x0 <- x0 - solve(hessian) %*% gradient

      # 检查收敛性
      if (max(abs(gradient)) < tol) {
        return(x0)
      }
    }
  }

  stop("Method did not converge in max_iter iterations!")
}
```

2 求根

2.1 $x^2 = 5$

```
f1 <- function(x) x^2 - 5
root1 <- newton_method(f1, x0=2) # 使用 2 作为初始猜测值
cat(" 该方程的根为: ",root1)
```

```
## 该方程的根为:  2.236068
```

2.2 $\sqrt{|x|}$

```
f2 <- function(x) sqrt(abs(x))
root2 <- newton_method(f2,x0 = 0.25)
```

```
## Error in D(body(f), "x"): 微分表里没有这个函数'abs'
```

```
cat(" 该方程的根为: ",root2)
```

```
## Error in eval(expr, envir, enclos): 找不到对象'root2'
```

报错显示 |x| 不存在导数
尝试将 $\sqrt{(abs(x))}$ 变得" 可微", 采用数值微分计算而不是用现成的 D(), 因为 R 程序包中的 D() 无法对非连续函数求导

```
newton_1 <- function(f, x0, tol=1e-6, max_iter=1000, numerical_diff=FALSE) {
  delta <- 1e-5 # 用于数值微分的小增量

  # 数值微分函数
  numerical_derivative <- function(f, x) {
    (f(x + delta) - f(x - delta)) / (2 * delta)
  }

  for (i in 1:max_iter) {
    # 根据选项计算导数
    if (numerical_diff) {
      df <- numerical_derivative(f, x0)
    } else {
      df_expr <- D(body(f), "x")
      df_func <- function(x) eval(df_expr)
      df <- df_func(x0)
    }

    # 更新 x0
    x0 <- x0 - f(x0) / df
    cat("Iteration", i, ": x0 =", x0, "\n")

    # 检查收敛性
    if (abs(f(x0)) < tol) {
      return(x0)
    }
  }
```

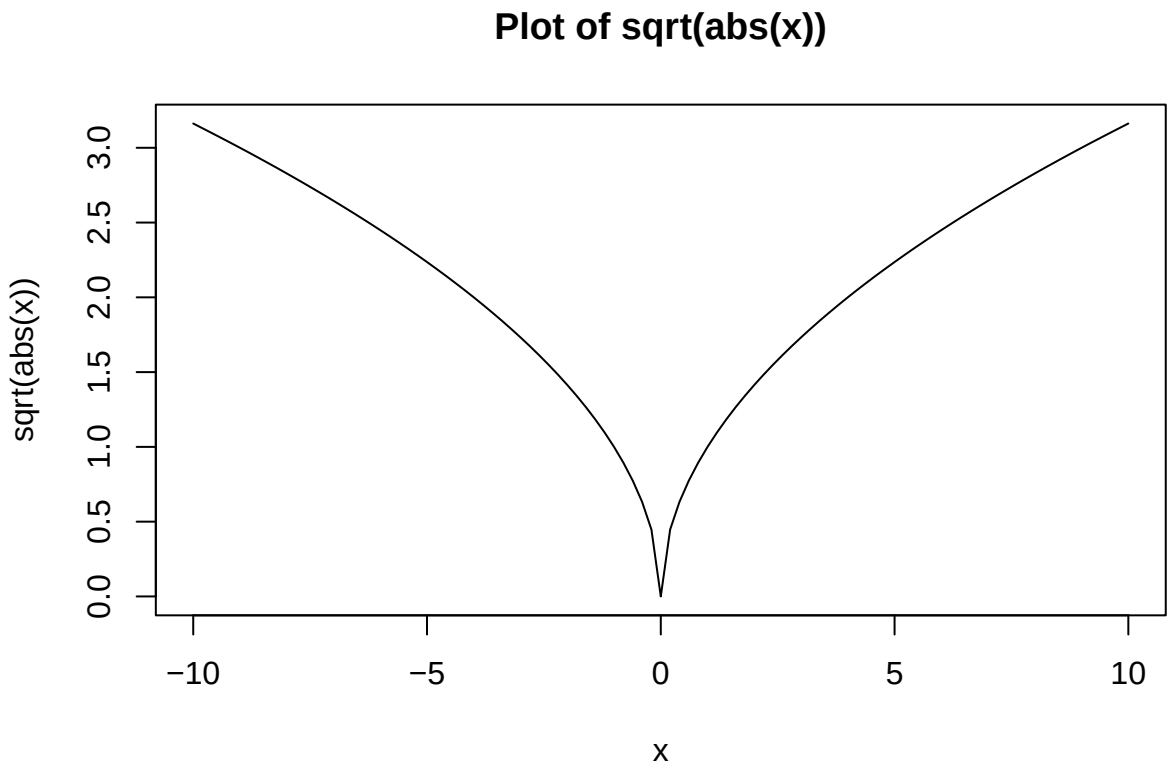
```
  stop("Method did not converge in max_iter iterations!")
}

# 示例
f2 <- function(x) {
  if (x >= 0) {
    return(sqrt(x))
  } else {
    return(sqrt(-x))
  }
}

root <- newton_1(f2, x0=0.25, max_iter=10,numerical_diff=TRUE)
```

```
## Iteration 1 : x0 = -0.25
## Iteration 2 : x0 = 0.25
## Iteration 3 : x0 = -0.25
## Iteration 4 : x0 = 0.25
## Iteration 5 : x0 = -0.25
## Iteration 6 : x0 = 0.25
## Iteration 7 : x0 = -0.25
## Iteration 8 : x0 = 0.25
## Iteration 9 : x0 = -0.25
## Iteration 10 : x0 = 0.25
```

```
## Error in newton_1(f2, x0 = 0.25, max_iter = 10, numerical_diff = TRUE): Method did not converge in ma
```

这次在微分阶段没有问题，但是使用牛顿法迭代无法得到收敛的结果。我们输出了迭代十次 x 的取值，发现其一直在 ± 0.25 间循环，绘制该函数图像有利于我们进一步了解该函数的性质。

```
# 使用 curve() 函数绘制 sqrt(abs(x))
curve(sqrt(abs(x)), from = -10, to = 10,
      main = "Plot of sqrt(abs(x))", xlab = "x", ylab = "sqrt(abs(x))")
```



$\sqrt{|x|}$ 在 0 点处不连续，根据牛顿法的几何解释每次迭代都与切线和 x 轴的交点有关。因此牛顿法无法求解该类函数的根，无法收敛。

2.3 $x \times e^{-x^2} - 0.4(e^x + 1)^{-1} - 0.2$

将起点设定为 0.5

```
f3 <- function(x) x * exp(-x^2) - 0.4 * (exp(x) + 1)^(-1) - 0.2

root3 <- newton_method(f3,x0 = 0.5)
cat(" 该方程的根为: ",root3)
```

```
## 该方程的根为:  0.4303876
```

将起点设定为 0.6

```
root3 <- newton_method(f3,x0 = 0.6)
cat(" 该方程的根为: ",root3)
```

```
## 该方程的根为:  0.4303876
```

不同的起点最后收敛的根值相同

2.4 Other examples

2.4.1 $x^3 - 2x^2 - 11x + 12$

```
f4 <- function(x) x^3-2*x^2-11*x+12
root4 <- newton_method(f4,x0=2.35287527)
cat(" 当 x0=2.35287527 时，该方程的根为：",root4)
```

```
## 当x0=2.35287527时，该方程的根为： 4
```

```
root4 <- newton_method(f4,x0=2.35284127)
cat(" 当 x0=2.35284127 时，该方程的根为：",root4)
```

```
## 当x0=2.35284127时，该方程的根为： -3
```

不同的初始值得到了不同的根，根本原因在于这个方程的根不唯一，从不同的初始点收敛会有不同的效果

2.4.2 $2x^3 + 3x^2 + 5$

```
f5 <- function(x) 2*x^3+3*x^2+5
cat(" 当 x0=0.5 时，该方程的根为",newton_method(f5,x0 = 0.5),"\n")
```

```
## 当x0=0.5时，该方程的根为 -2.078617
```

```
cat(" 当 x0=0 时，该方程的根为",newton_method(f5,x0 = 0.5))
```

```
## 当x0=0时，该方程的根为 -2.078617
```

不同的初始值收敛至相同的根

2.5 牛顿法求根总结

- 牛顿法求根于初始值有关，不同的初始值可能收敛至不同的根。（所以如果一个方程有多个根需要多尝试并挑选初始值）
- 如果一阶导数为零，Newton-Raphson 定理就不成立。
- 如果函数在根的邻域内不是连续可微的，则牛顿法可能总是发散或失败。

3 优化

牛顿法求极值的本质是使 $f'(x^*) = 0$，因此可以把牛顿法求根的方法拓展到对 $f'(x)$ 求根。即回到公式 $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$

应用：寻找 $-x^4$ 的最大值

```
f6 <- function(x) -x^4
df6_expr <- D(body(f6), "x")
df6 <- function(x) eval(df6_expr)
root6 <- newton_1(df6,x0=0.5,max_iter=100,numerical_diff = TRUE)
```

```
## Iteration 1 : x0 = 0.3333333
## Iteration 2 : x0 = 0.2222222
## Iteration 3 : x0 = 0.1481481
## Iteration 4 : x0 = 0.09876543
## Iteration 5 : x0 = 0.06584362
## Iteration 6 : x0 = 0.04389575
## Iteration 7 : x0 = 0.02926383
## Iteration 8 : x0 = 0.01950922
## Iteration 9 : x0 = 0.01300615
## Iteration 10 : x0 = 0.008670766
## Iteration 11 : x0 = 0.005780512
```

```
cat(" 当 x0=1 时,f(x) 的极大值点为: ",newton_1(df6,x0=0.5,max_iter=100,numerical_diff = TRUE))
```

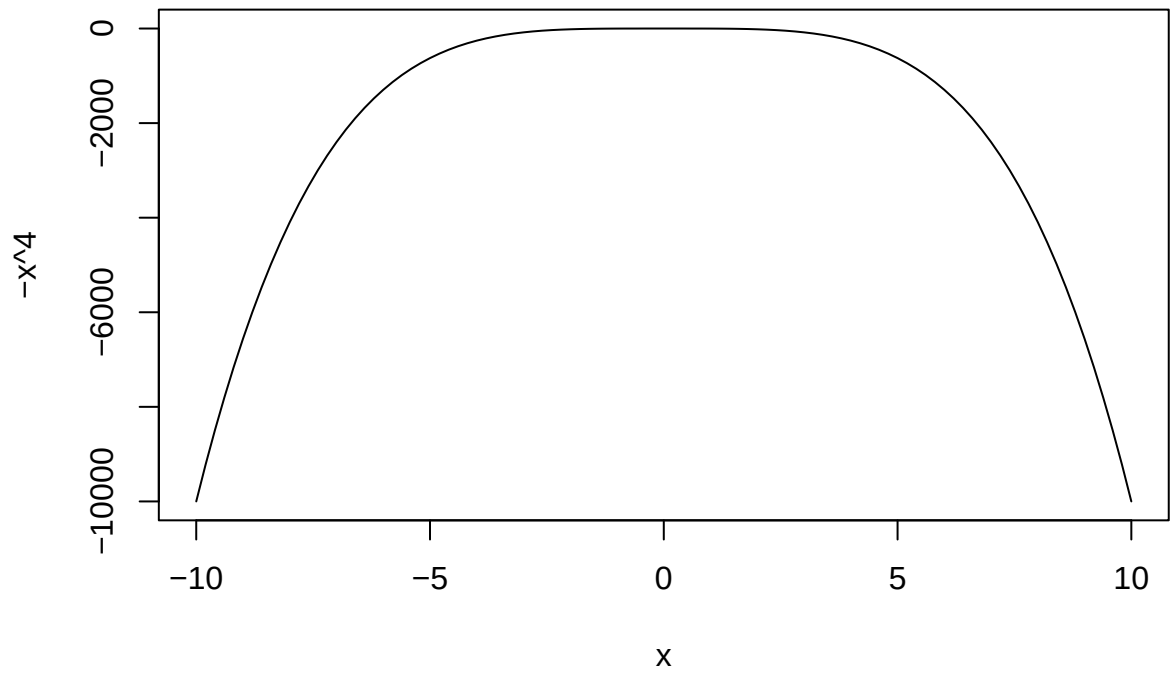
```
## Iteration 1 : x0 = 0.3333333
## Iteration 2 : x0 = 0.2222222
## Iteration 3 : x0 = 0.1481481
## Iteration 4 : x0 = 0.09876543
## Iteration 5 : x0 = 0.06584362
## Iteration 6 : x0 = 0.04389575
## Iteration 7 : x0 = 0.02926383
## Iteration 8 : x0 = 0.01950922
## Iteration 9 : x0 = 0.01300615
## Iteration 10 : x0 = 0.008670766
## Iteration 11 : x0 = 0.005780512
## 当x0=1时,f(x)的极大值点为： 0.005780512
```

```
cat(" 牛顿法得到的极大值为: ",f6(root6))
```

```
## 牛顿法得到的极大值为： -1.116517e-09
```

由于容忍度设定的原因，当迭代至 x0 为 0.0057 时截止.

```
curve(-x^4, from = -10, to = 10)
```



通过图像，

可知 $f(x)$ 的极大值在 x=0 时取到。

4 多元函数

牛顿法求多元函数极值可以使用以下公式：

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

```
f2 <- function(x1, x2) (x1)^2 + (x2)^2 - (x1)*(x2)+exp(x2)
root2 <- newton_method(f2, c(2, 1))
cat(" 该多元函数的极值点为",root2,"\n")
```

```
## 该多元函数的极值点为 -0.2162814 -0.4325628
```

```
extrema_value <- do.call(f2, as.list(root2))
cat(" 该多元函数的极值为: ",extrema_value)
```

```
## 该多元函数的极值为:  0.789177
```