hw 2 solution

Statistical Computing, Jieun Shin

Autumn 2021

문제 1.

1. 각 알고리즘 함수를 정의한다. (예제 코드 사용)

```
# 이분법
Bisection = function(x0, x1, epsilon = 1e-5)
{
    fx0 = f(x0)
    fx1 = f(x1)
    if (fx0 * fx1 >0)
        return("wrong initial values")
    error = abs(x1 - x0)
    N = 1
    x2 = (x0 + x1) / 2
    while (error > epsilon)
    {
        N = N + 1
        error = error / 2
        x2 = (x0 + x1) / 2
        fx2 = f(x2)
        if (fx0 * fx2 < 0)
        {
            x1 = x2; fx1 = fx2
        } else
        {
            x0 = x2; fx0 = fx2
        }
    }

    return(list(x = x2, n = N))
}

# 뉴턴법
Newton = function(x0, epsilon = 1e-5, n = 100)
{
    e = 1
    N = 1
    d = epsilon
    while (e > epsilon)
    {
        N = N + 1
        if (N > n)
            return("not converge after 100 iterations")
```

```
        x1 = x0 - f(x0) * d / (f(x0 + d) - f(x0))
        e = abs(x1 - x0)
        x0 = x1
    }
    return(list(x = x1, n = N))
}
```

2. 최소화 할 함수 $f(x) = -exp(-x) + 2x$를 정의한다.

```
f = function(x){-exp(-x) + 2*x}
```

3. 뉴턴법과 이분법을 비교한다.

```
Newton(1, n = 100)
```

```
## $x
## [1] 0.3517337
##
## $n
## [1] 5
```

```
Bisection(0, 1)
```

```
## $x
## [1] 0.351738
##
## $n
## [1] 18
```

뉴턴법과 이분법 모두 근사값 $x = 0.352$을 출력하였으며 뉴턴법은 5번, 이분법은 18번의 반복만에 도달하였다.

문제 2.

1. 함수 $f(x,y) = (x-y)^2 + (x-2)^2 + (y-3)^4$와 편도함수 $\nabla f = (2(x-y) + 2(x-2), -2(x-y) + 4(y-3)^3)$를 정의한다.

```
f = function(x)
{
  f = (x[1] - x[2])^2 + (x[1] - 2)^2 + (x[2] - 3)^4
}

df = function(x)
{
  df1 = 2 * (x[1] - x[2]) + 2 * (x[1] - 2)
  df2 = -2 * (x[1] - x[2]) + 4 * (x[2] - 3)^3
  df = c(df1, df2)
  return(df)
}
```

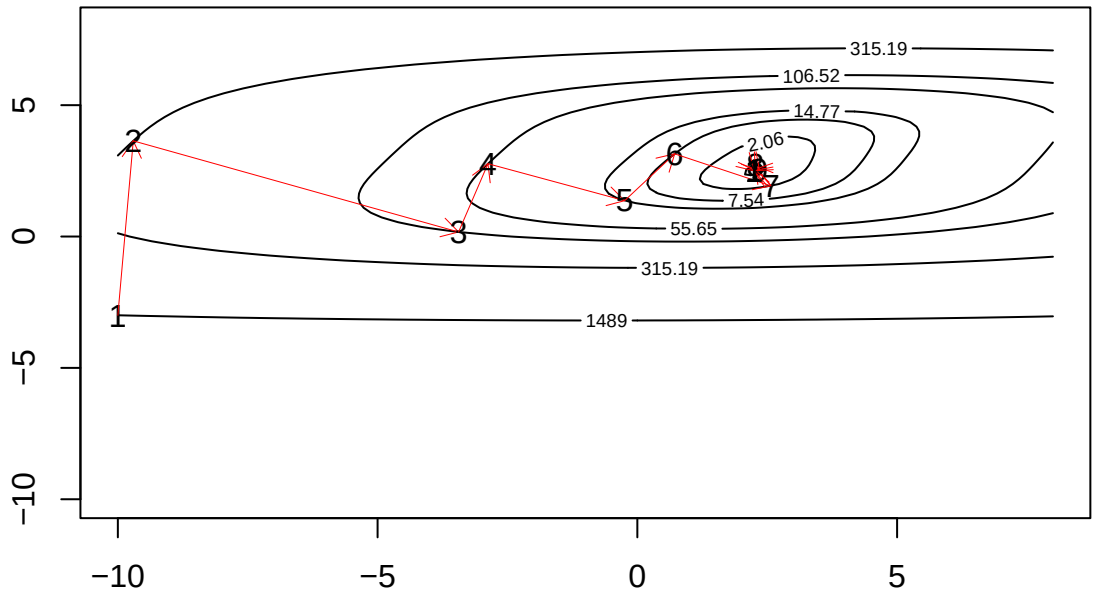
2. 최대하강법을 이용하여 최소점을 찾는다.

```
m = 100
x1 = x2 = seq(-10, 8, length=m)
xg = expand.grid(x1, x2)
z = matrix(apply(xg, 1, f), m, m)
xh = NULL; fh = NULL
x0 = c(-10, -3); fx0 = f(x0); ni = 0
```

```
eps = 1e-4
niter = 100

# 최대하강법 알고리즘
for (i in 1:niter)
{
  xh = rbind(xh, x0); fh = c(fh, fx0); ni = ni+1

  cat("iteration=", i)
  cat("  x0=", round(x0,2), "  f(x0)=", round(f(x0),3), "\n")

  d = df(x0)
  for (iters in 1:20)
  {
    x = x0 - d; fx = f(x)
    if (fx < fx0) break
    d = d / 2
  }

  # stopping rule
  if(abs(fx-fx0) < eps) break

  x0 = x; fx0 = fx
}
```

```
## iteration= 1  x0= -10 -3   f(x0)= 1489
## iteration= 2  x0= -9.7 3.64   f(x0)= 315.187
## iteration= 3  x0= -3.44 0.17   f(x0)= 106.525
## iteration= 4  x0= -2.88 2.77   f(x0)= 55.652
## iteration= 5  x0= -0.25 1.37   f(x0)= 14.775
## iteration= 6  x0= 0.72 3.15   f(x0)= 7.538
## iteration= 7  x0= 2.57 1.93   f(x0)= 2.055
## iteration= 8  x0= 2.27 2.7   f(x0)= 0.269
## iteration= 9  x0= 2.35 2.51   f(x0)= 0.206
## iteration= 10  x0= 2.26 2.55   f(x0)= 0.193
## iteration= 11  x0= 2.27 2.49   f(x0)= 0.189
## iteration= 12  x0= 2.25 2.51   f(x0)= 0.188
## iteration= 13  x0= 2.26 2.5   f(x0)= 0.188
## iteration= 14  x0= 2.25 2.5   f(x0)= 0.188
```

```
contour(x1, x2, z, levels = round(fh, 2))

#
for (i in 1:(ni-1))
{
  points(xh[i,1], xh[i,2], pch = as.character(i))
  x1=xh[i,1]; y1=xh[i,2]; x2=xh[i+1,1]; y2=xh[i+1,2]
  arrows(x1, y1, x2, y2, length=0.1, col="red", lwd=0.5)
}
points(xh[ni,1], xh[ni,2], pch=as.character(ni))
```



3. 뉴튼랩슨법을 이용하여 최소점을 찾는다. 이 때 이계도함수

$$\nabla^2 f = \begin{pmatrix} 4 & -2 \\ -2 & 12(y-3)^2 + 2 \end{pmatrix}$$

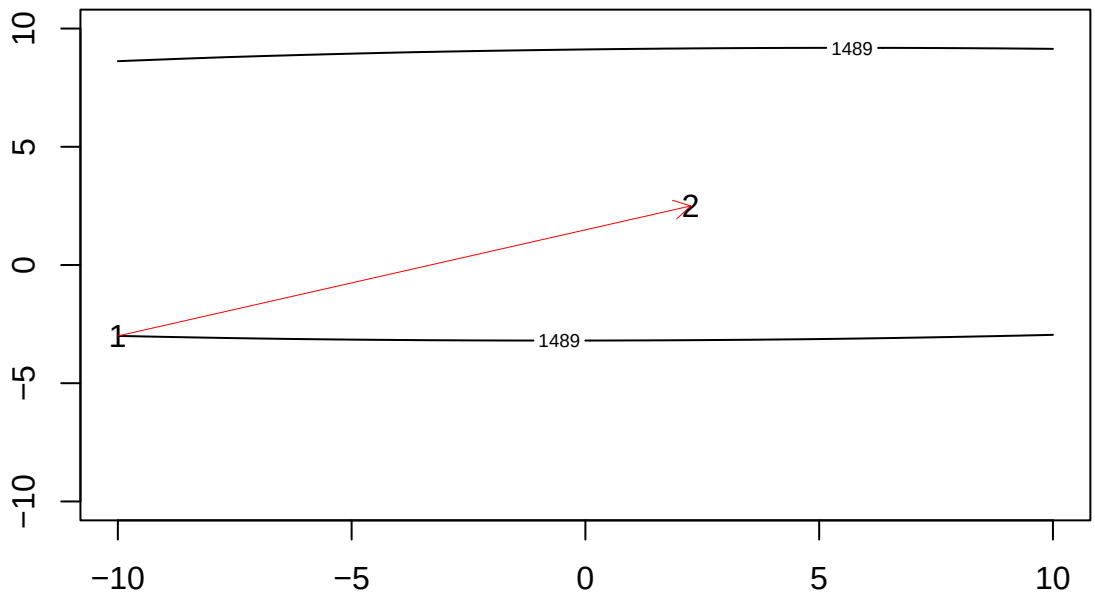
를 정의한다.

```
x1 = x2 = seq(-10, 10, length=m)
xg = expand.grid(x1, x2)
z = matrix(apply(xg, 1, f), m, m)
xh = NULL; fh = NULL
x0 = c(-10, -3); fx0 = f(x0); ni = 0

# 이계도함수 정의
df2 = matrix(c(4, -2, -2, 12*(x0[2] - 3)^3 - 2) ,2,2); v = solve(df2)

eps = 1e-4
niter = 100

# 뉴튼랩슨법 알고리즘
for (i in 1:niter)
{
  xh = rbind(xh, as.vector(x0)); fh = c(fh, fx0); ni = ni+1

  cat("iteration=", i)
  cat("  x0=", round(x0,2), "  f(x0)=", round(f(x0),3), "\n")
```

```
  d = v %*% df(x0)

  # stopping rule
  if (abs(fx-fx0) < eps) break

  x0 = x; fx0 = fx
}
```

```
## iteration= 1  x0= -10 -3   f(x0)= 1489
## iteration= 2  x0= 2.25 2.5   f(x0)= 0.188
```

```
contour(x1, x2, z, levels=round(fh, 2))
for (i in 1:(ni-1))
{
  points(xh[i,1], xh[i,2], pch=as.character(i))
  x1=xh[i,1]; y1=xh[i,2]; x2=xh[i+1,1]; y2=xh[i+1,2]
  arrows(x1, y1, x2, y2, length=0.1, col="red", lwd=0.5)
}
points(xh[ni,1], xh[ni,2], pch=as.character(ni))
```



최대하강법과 뉴튼랩슨법 모두 $x = (2.25, 2.5)$에서 $f(x) = 0.188$으로 수렴하지만 최대하강법은 23번의 반복 후 수렴값에 도달하였고, 뉴튼랩슨법은 2번만에 수렴값에 도달하였다. 이를 통해 뉴튼랩슨법의 수렴속도가 훨씬 빠름을 확인할 수 있다.

문제 3.

1. 수치적분을 위한 함수를 정의한다.

```
# 직사각형법
Integral = function(a, b, n)
{
    integral = 0
    h = (b - a) / n
    for (i in 1:n)
        integral = integral + h * f(a + (i-1/2) * h)

    return(integral)
}

# 사다리꼴법
Trapezoid = function(a, b, n = 50)
{
    h = (b - a) / n
    integral = (f(a) + f(b)) / 2

    x = a
    n1 = n - 1
    for (i in 1:n1)
    {
        x = x + h
        integral = integral + f(x)
    }
    integral = integral * h

    return(integral)
}

# 심슨 적분법
Simpson = function(a, b, n = 12)
{
    h = (b - a) / n
    integral = f(a) + f(b)
    x2 = a
    x3 = a + h
    even = 0
    odd = f(x3)
    h2 = 2 * h
    n1 = n / 2 - 1
    for (i in 1:n1)
    {
        x2 = x2 + h2
        x3 = x3 + h2
        even = even + f(x2)
        odd = odd + f(x3)
    }
    integral = (integral + 4 * odd + 2 * even) * h / 3

    return(integral)
}
```

2. 함수 $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ 정의 및 구간 수 $n = 2, 4, \ldots, 20$에 따른 각 수치적분 함수값을 비교한다.

```r
f = function(x) {exp(-x^2/2)/sqrt(2*pi)}
a = 0; b = 1
true = pnorm(1) - 0.5
n = seq(2, 20, length.out = 10)
for(i in n){
  int = Integral(a, b, i)
  trap = Trapezoid(a, b, i)
  sim = Simpson(a, b, i)

  cat("n = ", i, "참값 = ", round(true, 5), '\n',
      "직사각형법 = ", round(int, 5), ", 차이 =", round(int - true, 5), '\n',
      "사다리꼴법 = ", round(trap, 5), ", 차이 =", round(trap - true, 5), '\n',
      "심슨법 =", round(sim, 5), ", 차이 =", round(sim - true, 5), '\n', '\n')
}
```

```
## n =  2 참값 =  0.34134
##  직사각형법 =  0.3439 , 차이 = 0.00256
##  사다리꼴법 =  0.33626 , 차이 = -0.00508
##  심슨법 = 0.53821 , 차이 = 0.19687
##
## n =  4 참값 =  0.34134
##  직사각형법 =  0.34198 , 차이 = 0.00063
##  사다리꼴법 =  0.34008 , 차이 = -0.00126
##  심슨법 = 0.34136 , 차이 = 1e-05
##
## n =  6 참값 =  0.34134
##  직사각형법 =  0.34163 , 차이 = 0.00028
##  사다리꼴법 =  0.34078 , 차이 = -0.00056
##  심슨법 = 0.34135 , 차이 = 0
##
## n =  8 참값 =  0.34134
##  직사각형법 =  0.3415 , 차이 = 0.00016
##  사다리꼴법 =  0.34103 , 차이 = -0.00032
##  심슨법 = 0.34135 , 차이 = 0
##
## n =  10 참값 =  0.34134
##  직사각형법 =  0.34145 , 차이 = 1e-04
##  사다리꼴법 =  0.34114 , 차이 = -2e-04
##  심슨법 = 0.34135 , 차이 = 0
##
## n =  12 참값 =  0.34134
##  직사각형법 =  0.34141 , 차이 = 7e-05
##  사다리꼴법 =  0.3412 , 차이 = -0.00014
##  심슨법 = 0.34134 , 차이 = 0
##
## n =  14 참값 =  0.34134
##  직사각형법 =  0.3414 , 차이 = 5e-05
##  사다리꼴법 =  0.34124 , 차이 = -1e-04
##  심슨법 = 0.34134 , 차이 = 0
##
## n =  16 참값 =  0.34134
##  직사각형법 =  0.34138 , 차이 = 4e-05
##  사다리꼴법 =  0.34127 , 차이 = -8e-05
##  심슨법 = 0.34134 , 차이 = 0
```

```
##
## n =  18 참값 =  0.34134
##  직사각형법 =  0.34138 , 차이 = 3e-05
##  사다리꼴법 =  0.34128 , 차이 = -6e-05
##  심슨법 = 0.34134 , 차이 = 0
##
## n =  20 참값 =  0.34134
##  직사각형법 =  0.34137 , 차이 = 3e-05
##  사다리꼴법 =  0.34129 , 차이 = -5e-05
##  심슨법 = 0.34134 , 차이 = 0
##
```

결과를 보면 직사각형법과 사다리꼴법에 비해 심슨법이 참값과의 차이가 가장 작음을 확인할 수 있다.