hw 2 solution

Statistical Computing, Jieun Shin

2022-10-11

문제 1.

$x^3 - 10x^2 + 5 = 0$의 근은 구간 (0,1)상에 있다. 초기값 $x_0 = 1$인 뉴턴법과 초기구간 [0,1]인 이분법을 이용하여 오차한계 $\epsilon = 1e-4$을 만족하는 근을 구하시오. 두 알고리즘에 대해서 k (반복수)와 x_k (근사적인 해)를 출력하고 비교하시오 (10점).

1. 각 알고리즘 함수를 정의한다.

```
# 이분법
Bisection = function(x0, x1, epsilon = 1e-4)
{
  x = sort(runif(100, 0, 1))
  fx = f(x)
  plot(x, fx, type = 'l')
  abline(h = 0)

  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
      }
        text(x2, f(x2), labels = N, col = 'red')
  }

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

# 뉴턴법
Newton = function(x0, epsilon = 1e-4, n = 100)
```

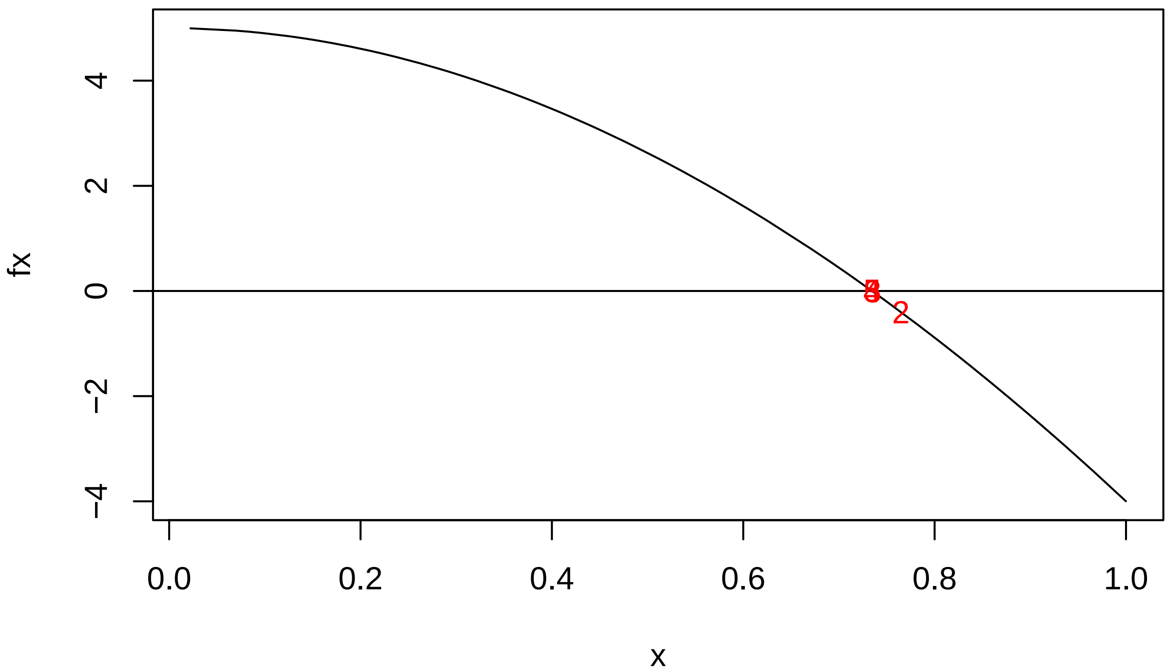
```
{
  x = sort(runif(100, 0, 1))
  fx = f(x)
  plot(x, fx, type = 'l')
  abline(h = 0)

  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

      text(x1, f(x1), labels = N, col = 'red')
  }
  return(list(x = x1, iter = N))
}
```

2. 최소화 할 함수 $f(x) = x^3 - 10x^2 + 5$를 정의한다.

```
f = function(x){x^3 - 10 *x^2 + 5}
```

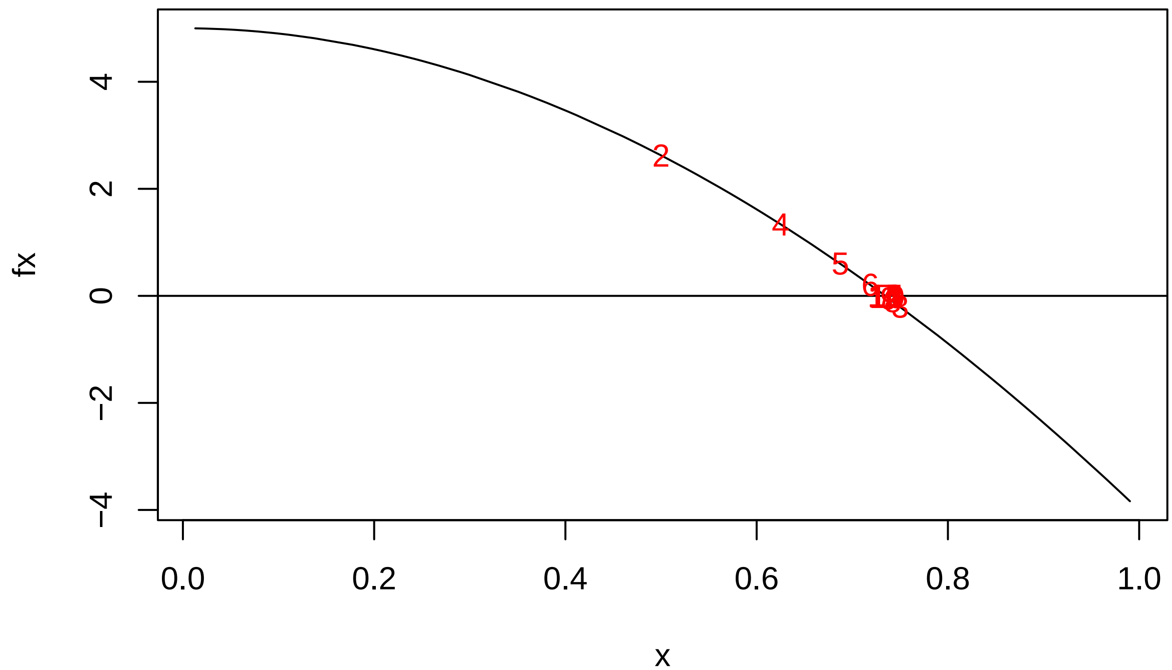
3. 뉴턴법과 이분법을 비교한다.

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




```
## $x
## [1] 0.7346035
##
## $iter
## [1] 5
```

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




```
## $x
## [1] 0.7345581
##
## $iter
## [1] 15
```

뉴턴법과 이분법 모두 근사값 $x = 0.734$을 출력하였으며 뉴턴법은 5번, 이분법은 15번의 반복만에 도달하였다.

문제 2.

함수 $f(x, y) = 100(y - x^2)^2 + (1 - x)^2$의 최소점과 최소값을 초기치 $x_0 = -1, y_0 = 1$인 최대하강법과 뉴튼-랩슨 알고리즘을 이용하여 구하시오. 단 오차한계는 $\epsilon = 1e - 4$이다 (10점).

1. 함수 $f(x, y) = 100(y - x^2)^2 + (1 - x)^2$와 편도함수 $\nabla f = (-400x(y - x^2) - 2(1 - x), 200(y - x^2))$를 정의한다.

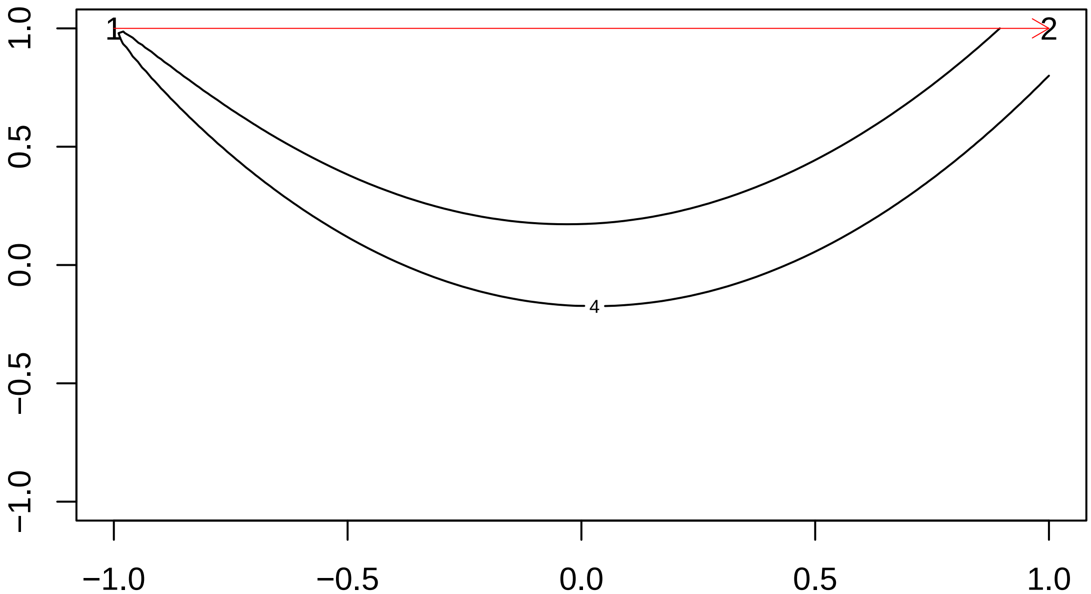
```
f = function(x)
{
  100*(x[2] - x[1]^2)^2 + (1-x[1])^2
}

df = function(x)
{
  df1 = -400*x[1]*(x[2] - x[1]^2) - 2*(1-x[1])
  df2 = 200*(x[2]-x[1]^2)
  df = c(df1, df2)
  return(df)
}
```

2. 최대하강법을 이용하여 최소점을 찾는다.

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

eps = 1e-4
max_iter = 100
for (i in 1:max_iter)
{
  xh = rbind(xh, x0); fh = c(fh, fx0); ni = ni+1
  cat("iteration=", round(i,2))
  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= -1 1   f(x0)= 4
## iteration= 2  x0= 1 1   f(x0)= 0
```

```r
# contour
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

$$\nabla^2 f = \begin{pmatrix} -400(y - 3x^2) + 2 & -400x \\ -400x & 200 \end{pmatrix}$$

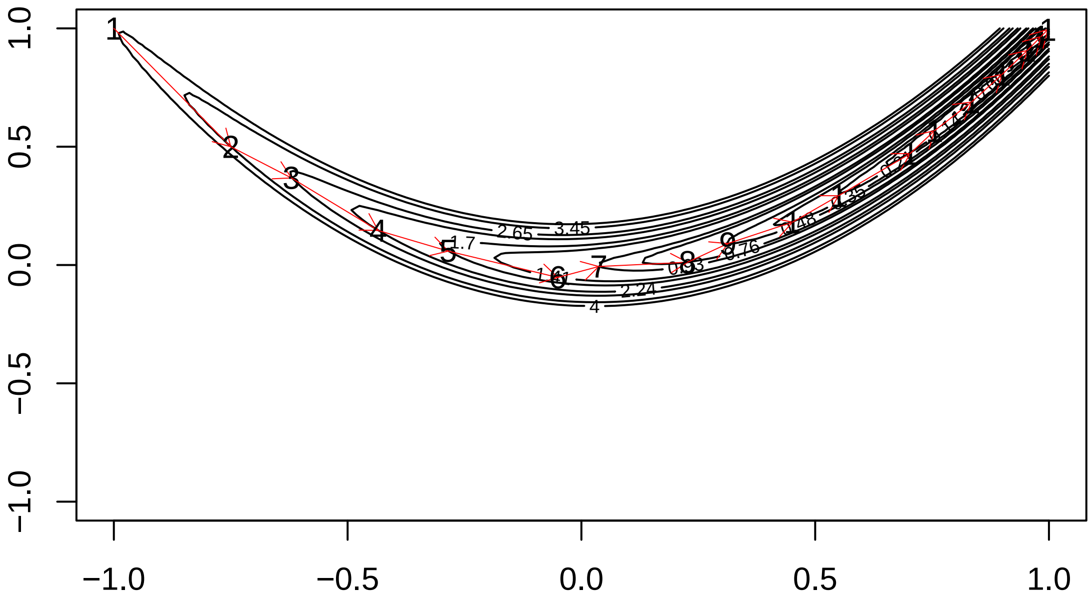
를 정의한다.

```
m=100
x1 = x2 = seq(-1, 1, length=m)
xg = expand.grid(x1, x2)
z = matrix(apply(xg, 1, f), m, m)
xh = NULL; fh = NULL
x0 = c(-1, 1); fx0 = f(x0); ni = 0
df2 = function(x){
  matrix(c(-400*(x0[2]-3*x0[1]^2) + 2,
              -400*x0[1],
              -400*x0[1],
              200)
            , 2, 2)
  }

eps = 1e-4
max_iter = 100

for (i in 1:max_iter)
{
  xh = rbind(xh, as.vector(x0)); fh = c(fh, fx0); ni = ni+1
  cat("iteration=", round(i,2))
```

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

  v = solve(df2(x0))
  d = v %*% 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= -1 1   f(x0)= 4
## iteration= 2  x0= -0.75 0.5   f(x0)= 3.453
## iteration= 3  x0= -0.62 0.37   f(x0)= 2.654
## iteration= 4  x0= -0.43 0.15   f(x0)= 2.242
## iteration= 5  x0= -0.28 0.06   f(x0)= 1.701
## iteration= 6  x0= -0.05 -0.05   f(x0)= 1.405
## iteration= 7  x0= 0.04 -0.01   f(x0)= 0.933
## iteration= 8  x0= 0.23 0.01   f(x0)= 0.757
## iteration= 9  x0= 0.31 0.09   f(x0)= 0.477
## iteration= 10  x0= 0.45 0.18   f(x0)= 0.353
## iteration= 11  x0= 0.55 0.29   f(x0)= 0.212
## iteration= 12  x0= 0.7 0.47   f(x0)= 0.143
## iteration= 13  x0= 0.76 0.57   f(x0)= 0.061
## iteration= 14  x0= 0.83 0.69   f(x0)= 0.033
## iteration= 15  x0= 0.9 0.81   f(x0)= 0.012
## iteration= 16  x0= 0.95 0.91   f(x0)= 0.003
## iteration= 17  x0= 0.98 0.97   f(x0)= 0
## iteration= 18  x0= 1 0.99   f(x0)= 0
```

```r
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 $(x, y) = (1, 1)$에서 $f(x) = 0$으로 수렴하였다.

문제 3.

적분값 $\int_0^\pi \sim (x)dx$을 직사각형법, 사다리꼴법, 심슨법을 이용하여 구하고자 한다. 각 알고리즘에 대하여 $n(= 2, 3, \ldots, 20)$ (구간 수), I_n (적분 값), 그리고 참값 2와의 차이 $|I_n - 2|$을 출력하시오 (10점).

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 $\sin(x)$ 정의 및 구간 수 $n = 2, 4, \dots, 20$에 따른 각 수치적분 함수값을 비교한다.

```
f = function(x) {sin(x)}
a = 0; b = pi
true = 2
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 =  2
##  직사각형법 =  2.22144 , 차이 = 0.22144
##  사다리꼴법 =  1.5708  , 차이 = -0.4292
##  심슨법 = 2.0944 , 차이 = 0.0944
```

```
##
## n =  4 참값 =  2
##  직사각형법 =  2.05234 , 차이 = 0.05234
##  사다리꼴법 =  1.89612 , 차이 = -0.10388
##  심슨법 = 2.00456 , 차이 = 0.00456
##
## n =  6 참값 =  2
##  직사각형법 =  2.02303 , 차이 = 0.02303
##  사다리꼴법 =  1.9541 , 차이 = -0.0459
##  심슨법 = 2.00086 , 차이 = 0.00086
##
## n =  8 참값 =  2
##  직사각형법 =  2.01291 , 차이 = 0.01291
##  사다리꼴법 =  1.97423 , 차이 = -0.02577
##  심슨법 = 2.00027 , 차이 = 0.00027
##
## n =  10 참값 =  2
##  직사각형법 =  2.00825 , 차이 = 0.00825
##  사다리꼴법 =  1.98352 , 차이 = -0.01648
##  심슨법 = 2.00011 , 차이 = 0.00011
##
## n =  12 참값 =  2
##  직사각형법 =  2.00572 , 차이 = 0.00572
##  사다리꼴법 =  1.98856 , 차이 = -0.01144
##  심슨법 = 2.00005 , 차이 = 5e-05
##
## n =  14 참값 =  2
##  직사각형법 =  2.0042 , 차이 = 0.0042
##  사다리꼴법 =  1.9916 , 차이 = -0.0084
##  심슨법 = 2.00003 , 차이 = 3e-05
##
## n =  16 참값 =  2
##  직사각형법 =  2.00322 , 차이 = 0.00322
##  사다리꼴법 =  1.99357 , 차이 = -0.00643
##  심슨법 = 2.00002 , 차이 = 2e-05
##
## n =  18 참값 =  2
##  직사각형법 =  2.00254 , 차이 = 0.00254
##  사다리꼴법 =  1.99492 , 차이 = -0.00508
##  심슨법 = 2.00001 , 차이 = 1e-05
##
## n =  20 참값 =  2
##  직사각형법 =  2.00206 , 차이 = 0.00206
##  사다리꼴법 =  1.99589 , 차이 = -0.00411
##  심슨법 = 2.00001 , 차이 = 1e-05
##
```

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