HW4

John Rothen

9/26/2020

Homework 4

3.5.1

1

$f(x;t) = 1/(pi(1+(x-t)^2)$ -> L(t) = product(f(x;t)) = $pi^{-n} - Product_i^n(1+(t-x)^2)$. Thus, log(L(t))= l(t) = $-nln(pi) - Sum(ln(1+(t-x)^2))$. To obtain the first derivative, the -nln(pi) is removed as it equals =0, and ln(1 +(t-x)^2) -> (t-x)/(1+ (t-x)^2) via the power and chain rules. This can be done again to obtain the second derivative. The information can then be found be evaluating $I(t) = -E(l"(t)|t).

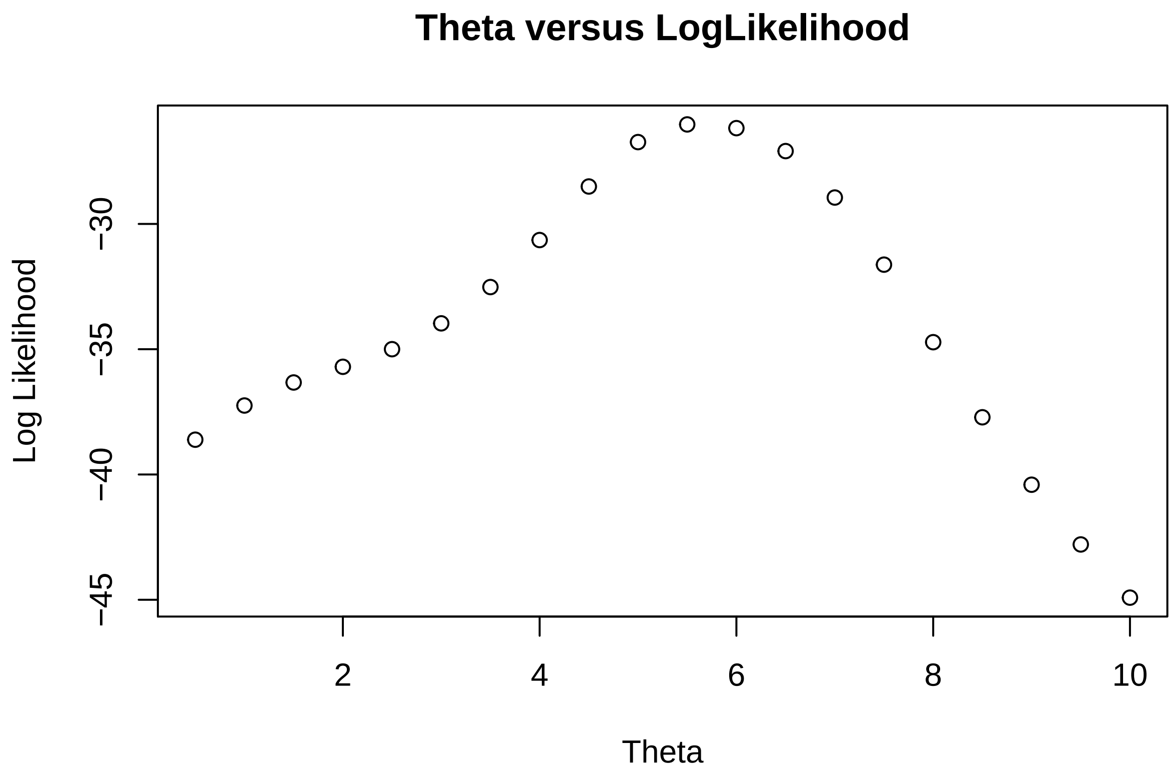
2

```r
set.seed(909)

sampl<-rcauchy(10,location=5, scale=1) #scale is 1 for this problem


loglikCauchy <- function(x,theta, n=10){
  #x is a vector of values
  part <- sum(log(1+(theta - x)^2))
  -n*log(pi) - part
}
theta.set <- seq(0.5,10, by=.5)
y <- rep(0,20)

for(i in 1:20){
  y[i] <- loglikCauchy(sampl, i/2)
}
theta.set <- c(theta.set)
plot(theta.set, y, xlab= "Theta", ylab = "Log Likelihood", main= "Theta versus LogLikelihood")
```



3

```
#loglik function
loglikCauchy <- function(x){
  part <- sum(log(1+(x - sampl)^2))
  -10*log(pi) - part
}
#first derivative of loglik
l2.c <- function(t){
  -2*sum( (t-sampl)/ (1 + (t -sampl) ^2) )
}

#root via newton-taphson
uniroot(l2.c, c(-10,30))
```

```
## $root
## [1] 5.6357152455319985
##
## $f.root
## [1] -3.5072528556390203e-07
##
## $iter
## [1] 10
##
## $init.it
## [1] NA
```

```
##
## $estim.prec
## [1] 6.4550742709279518e-05
```

It appears that the MLE of theta is about 5.5.

4

To improve the process, we can attempt to use M = loglikelihood's second derivative for the function $x_{t+1} = x_t - M_t^{-1} l'(x)$.

```
# second derivative of loglik
l3 <- function(x){
  -2*sum( ( (1 -(x-sampl)^2) / ((1+(x-sampl)^2)^2)))
}

#process
x=-10
for(i in 1:1000){
  x <- x + (l2.c(x))/(l3(x))
}
x
```

```
## [1] 5.2673645482712184
```

The results for this method do seem rather accurate, as it estimate the MLE to be theta.

5

The results of a univariate fixed-point iteration are shown below.

```
#Univariate Options
g1 <-function(t){
  1*l2.c(t) + t
}
g2 <-function(t){
  .64*l2.c(t) + t
}
g3 <-function(t){
  .25*l2.c(t) + t
}

#a=1
t=5
for(i in 1:100){
   t<-g1(t)
}
t
```

```
## [1] 5.3125345381118567
```

```
#a=.64
t=5
for(i in 1:100){
   t<-g2(t)
}
t
```

```
## [1] 3.8985495176085956
```

```r
#a=.25
t=5
for(i in 1:100){
   t<-g3(t)
}
t
```

```
## [1] 5.6357151165410073
```

The iterative methods seem efficient at alpha=1, and alpha=.25, as both appear to converge to the true MLE for theta. For alpha=.64, it appears that the iteration is not convergent to the MLE.

For a fixed-point Newton-Raphson method, one can use M = loglikelihood's second derivative of a good guess of the MLE (or the initial value), such as M = l"(5). The results are given below.

```r
x=-10
M = l3(5)
for(i in 1:100){
  x <- x - (l2.c(x))/M
}
x
```

```
## [1] 5.6357151165410064
```

This method provides an accurate estimate of the MLE, as seen in the earlier evaluations.

6

To use Fisher Scoring, we will use the form M = I_n = n/2

```r
#MLE via fisher scoring
x=-10
M= -(10/2)
for(i in 1:100){
  x <- x - (l2.c(x))/M
}
x
```

```
## [1] 5.6357151165410064
```

```r
#MLE further refined via Newton-Raphson without hessian using M=l''(x)
for(i in 1:100){
  x <- x- (l2.c(x))/(l3(x))
}
x
```

```
## [1] 5.6357151165410073
```

The final estimate is 5.63571511654101

7

Many of the methods used above find similar results. For accuracy, the Fisher scoring followed by Newton-Raphson refining is likely the best, as it refined the estimate via two methods. This method takes the longest though, as it takes the evaluation of the information. The multivariate fixed-point iteration method appeared to be the easiest method to perform, as it only required a rough estimate of the MLE to provide a very accurate estimate (virtually the same as is seen in the Fisher scoring version).

It is also worth noting that the method used in part 4 of this question provided the result closest to 5 (the original parameter used).

3.5.2

```r
x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2
n = length(x)
fn <- function(t){
  (1-cos(x-t))/(2*pi)
}
```

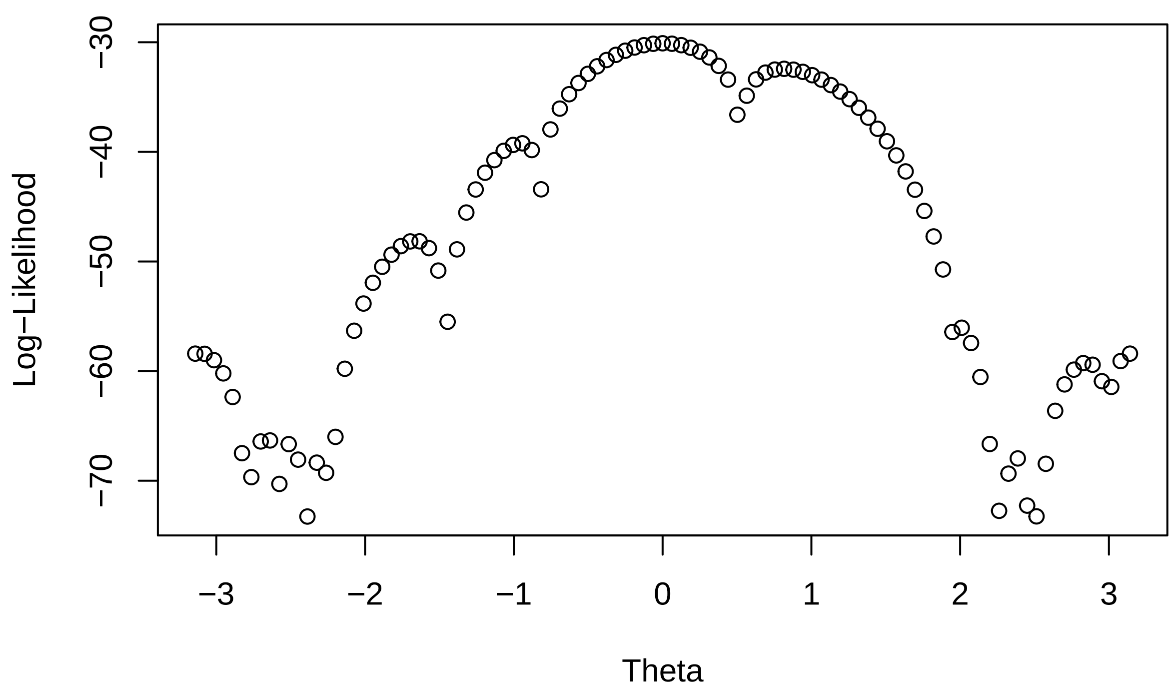
1

The likelihood function would be $ L(t) = Product_{i}^{n}[1-cos(x-t)]/ (2pi)^n $. Taking the log of this gives us $l(t) = sum(log(1 - cos(x - t))) - nln(2pi)$

```r
ll <-function(t){
  sum(log( 1-cos(x-t))) - (n*log(2*pi))
}
set <- c(seq(-pi, pi, by=0.06283185307179586 2))
lly <- rep(0,101)
for(i in 1:101){

  lly[i] <- ll(set[i])
}

plot(set, lly, xlab = "Theta", ylab="Log-Likelihood", )
```



2

Note: t is the short hand for theta.

$E(x|t) = integral(x * (1 - cos(x - t))/(2pi)) = -x((2sin(x - t) - x) + 2cos(x - t))/4pi$ from 2pi to 0 = sin(t) + pi = Xbar. From here we can further say that sin(t) = xbar-pi, and then $theta^{hat}$ = arcsin(xbar-pi). Because we can calculate xbar from the sample, the MOM estimate of Theta is given as:

```
xbar <- mean(x)
t.mom = asin(xbar - pi)
t.mom
```

```
## [1] 0.095394067305843627
```

Note: this value does appear to be close to the max likelihood theta from the graph in part 1.

3

```
ll1 <-function(t){
  sum(((sin(x-t)) / (1- cos(x-t))))
}

ll2 <- function(t){
  sum( ( -(sin(x-t)^2) +(cos(x-t))^2 -(cos(x-t)))/((cos(x-t)-1)^2))
}
#newton-raphson with fixed point l''(tmom) by hand
tnew <- t.mom
for (i in 1:20){
  tnew <- tnew - ((ll1(tnew))/ ll2(tnew))
}
tnew
```

```
## [1] 0.51983029708841644
```

```
#using general optim
optim(t.mom,ll)$par
```

```
## [1] 0.52000001809650964
```

The MLE appears to be around .52, and varies depending on methods.

4

```
#at -2.7
optim(-2.7, ll)$par
```

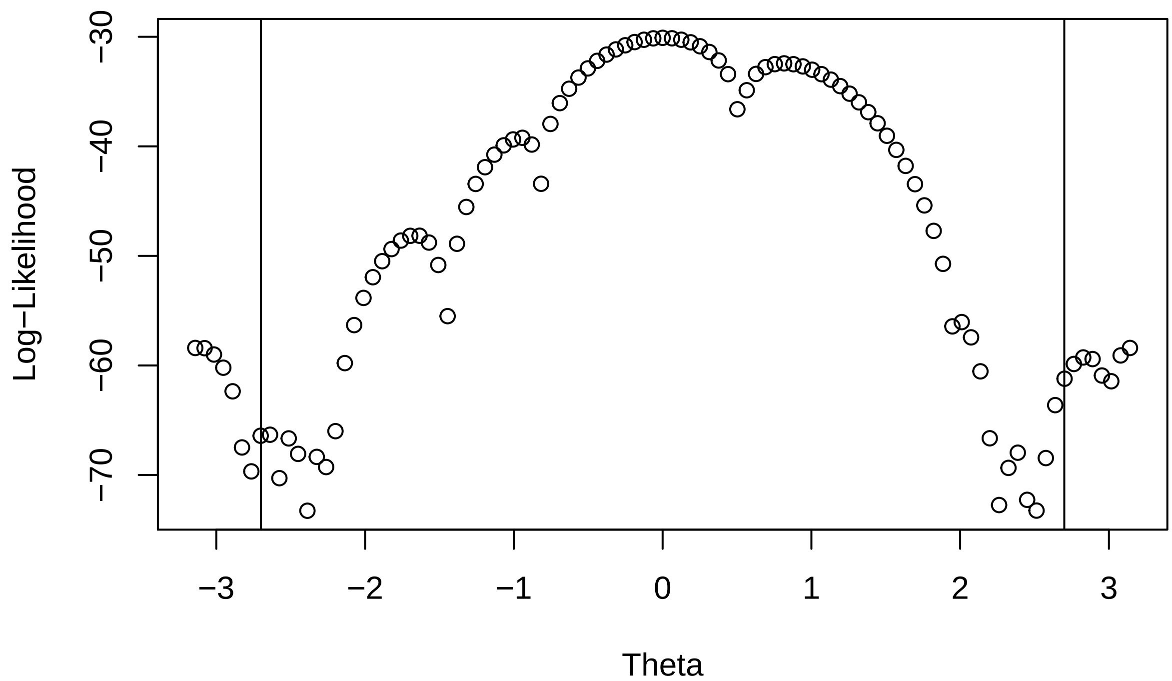
```
## [1] -2.3731852930784232
```

```
#at 2.7
optim(2.7,ll)$par
```

```
## [1] 2.9899999840557578
```

Both estimates are vastly different from the MLE estimate obtained in part 3. This is due to the existence of many local maximums. This can be seen on the graph, this time provided with verticals lines at both 2.7 and -2.7, which have local maximums closer to them then the ultimate maximum near t=0.1.

```
plot(set, lly, xlab = "Theta", ylab="Log-Likelihood", )
abline(v=2.7)
abline(v=-2.7)
```



5

The estimated zero for each given starting point is given in the following table as "output". These outputs were then placed into a group corresponding with the output, and placed within a group wih inputs that produced the same output. The levels (possible zeroes), are given in the output below, as well as the full print of each input/output and the correponding group they received. Group is numbered with 1 correponding to the lowest output (-2.814), and the next smallest being group 2, and so on.

```
range <- seq(-pi, pi, length.out= 200)
range <- c(range)

input <- range
output<- rep(0,200)
for(i in 1:200){
  temp <- input[i]
  output[i] <- optim(temp,ll)$par
}
dat <-cbind(input,output)
dat <- as.data.frame(dat)
options(digits=5)
dat$round <- round(dat$output,digits=3)
dat$group <- factor(dat$round)
levels(dat$group)
```

```
##  [1] "-2.813" "-2.753" "-2.583" "-2.463" "-2.403" "-2.373" "-2.243" "-2.223"
##  [9] "-1.463" "-1.433" "-0.823" "-0.821" "0.52"   "2.22"   "2.28"   "2.46"
## [17] "2.53"   "2.54"   "2.99"   "3.53"   "3.7"    "3.88"   "3.91"
```

```r
levels(dat$group) <- 1:23
fin<-dat[,c(1,2,4)]
fin
```

```
##         input   output group
## 1   -3.141593 -2.75319     2
## 2   -3.110019 -2.75319     2
## 3   -3.078445 -2.81319     1
## 4   -3.046871 -2.58319     3
## 5   -3.015297 -2.37319     6
## 6   -2.983724 -2.40319     5
## 7   -2.952150 -2.40319     5
## 8   -2.920576 -2.37319     6
## 9   -2.889002 -2.81319     1
## 10  -2.857428 -2.37319     6
## 11  -2.825855 -2.75319     2
## 12  -2.794281 -2.75319     2
## 13  -2.762707 -2.75319     2
## 14  -2.731133 -2.58319     3
## 15  -2.699560 -2.37319     6
## 16  -2.667986 -2.40319     5
## 17  -2.636412 -2.37319     6
## 18  -2.604838 -2.37319     6
## 19  -2.573264 -2.58319     3
## 20  -2.541691 -2.40319     5
## 21  -2.510117 -2.40319     5
## 22  -2.478543 -2.24319     7
## 23  -2.446969 -2.40319     5
## 24  -2.415395 -2.40319     5
## 25  -2.383822 -2.40319     5
## 26  -2.352248 -2.58319     3
## 27  -2.320674 -2.37319     6
## 28  -2.289100 -2.40319     5
## 29  -2.257526 -2.37319     6
## 30  -2.225953 -2.24319     7
## 31  -2.194379 -2.40319     5
## 32  -2.162805 -2.40319     5
## 33  -2.131231 -2.22319     8
## 34  -2.099657 -2.37319     6
## 35  -2.068084 -2.40319     5
## 36  -2.036510 -2.24319     7
## 37  -2.004936 -2.40319     5
## 38  -1.973362 -2.37319     6
## 39  -1.941788 -2.40319     5
## 40  -1.910215 -2.40319     5
## 41  -1.878641 -2.24319     7
## 42  -1.847067 -2.40319     5
## 43  -1.815493 -2.40319     5
## 44  -1.783919 -2.40319     5
## 45  -1.752346 -1.46319     9
## 46  -1.720772 -1.46319     9
## 47  -1.689198 -1.43319    10
## 48  -1.657624 -1.43319    10
## 49  -1.626050 -1.46319     9
```

```
## 50  -1.594477 -1.43319    10
## 51  -1.562903 -1.46319     9
## 52  -1.531329 -1.43319    10
## 53  -1.499755 -1.46319     9
## 54  -1.468181 -1.43319    10
## 55  -1.436608 -1.43319    10
## 56  -1.405034 -1.46319     9
## 57  -1.373460 -1.46319     9
## 58  -1.341886 -1.46319     9
## 59  -1.310313 -1.46319     9
## 60  -1.278739 -1.43319    10
## 61  -1.247165 -1.43319    10
## 62  -1.215591 -1.46319     9
## 63  -1.184017 -1.43319    10
## 64  -1.152444 -1.46319     9
## 65  -1.120870 -2.24319     7
## 66  -1.089296 -1.46319     9
## 67  -1.057722 -1.46319     9
## 68  -1.026148 -1.46319     9
## 69  -0.994575 -0.82319    11
## 70  -0.963001 -0.82319    11
## 71  -0.931427 -0.82319    11
## 72  -0.899853 -0.82319    11
## 73  -0.868279 -0.82319    11
## 74  -0.836706 -0.82319    11
## 75  -0.805132 -0.82319    11
## 76  -0.773558 -0.82319    11
## 77  -0.741984 -0.82319    11
## 78  -0.710410 -0.82319    11
## 79  -0.678837 -0.82319    11
## 80  -0.647263 -2.58319     3
## 81  -0.615689 -0.82319    11
## 82  -0.584115 -0.82319    11
## 83  -0.552541 -2.40319     5
## 84  -0.520968 -0.82319    11
## 85  -0.489394 -0.82319    11
## 86  -0.457820 -1.46319     9
## 87  -0.426246 -0.82319    11
## 88  -0.394672 -0.82319    11
## 89  -0.363099 -1.43319    10
## 90  -0.331525 -0.82319    11
## 91  -0.299951 -2.40319     5
## 92  -0.268377 -0.82319    11
## 93  -0.236803 -2.37319     6
## 94  -0.205230 -0.82319    11
## 95  -0.173656 -2.22319     8
## 96  -0.142082 -0.82319    11
## 97  -0.110508 -2.40319     5
## 98  -0.078934 -0.82092    12
## 99  -0.047361 -2.46276     4
## 100 -0.015787 -0.82092    12
## 101  0.015787  0.52000    13
## 102  0.047361  0.52000    13
## 103  0.078934  0.52000    13
```

```
## 104  0.110508  0.52000    13
## 105  0.142082  0.52000    13
## 106  0.173656  0.52000    13
## 107  0.205230  0.52000    13
## 108  0.236803  0.52000    13
## 109  0.268377  0.52000    13
## 110  0.299951  0.52000    13
## 111  0.331525  0.52000    13
## 112  0.363099  0.52000    13
## 113  0.394672  0.52000    13
## 114  0.426246  0.52000    13
## 115  0.457820  0.52000    13
## 116  0.489394  0.52000    13
## 117  0.520968  0.52000    13
## 118  0.552541  0.52000    13
## 119  0.584115  0.52000    13
## 120  0.615689  0.52000    13
## 121  0.647263  0.52000    13
## 122  0.678837  0.52000    13
## 123  0.710410  0.52000    13
## 124  0.741984  0.52000    13
## 125  0.773558  0.52000    13
## 126  0.805132  2.99000    19
## 127  0.836706  3.91000    23
## 128  0.868279  2.54000    18
## 129  0.899853  2.28000    15
## 130  0.931427  2.54000    18
## 131  0.963001  2.28000    15
## 132  0.994575  2.28000    15
## 133  1.026148  2.46000    16
## 134  1.057722  2.28000    15
## 135  1.089296  2.28000    15
## 136  1.120870  2.53000    17
## 137  1.152444  2.28000    15
## 138  1.184017  2.53000    17
## 139  1.215591  2.53000    17
## 140  1.247165  2.54000    18
## 141  1.278739  2.28000    15
## 142  1.310313  2.53000    17
## 143  1.341886  2.28000    15
## 144  1.373460  2.54000    18
## 145  1.405034  2.53000    17
## 146  1.436608  2.28000    15
## 147  1.468181  2.53000    17
## 148  1.499755  2.54000    18
## 149  1.531329  2.22000    14
## 150  1.562903  2.46000    16
## 151  1.594477  2.53000    17
## 152  1.626050  2.28000    15
## 153  1.657624  2.54000    18
## 154  1.689198  2.54000    18
## 155  1.720772  2.28000    15
## 156  1.752346  2.28000    15
## 157  1.783919  2.46000    16
```

```
## 158  1.815493  2.53000    17
## 159  1.847067  2.28000    15
## 160  1.878641  2.28000    15
## 161  1.910215  2.28000    15
## 162  1.941788  2.53000    17
## 163  1.973362  2.54000    18
## 164  2.004936  2.28000    15
## 165  2.036510  2.54000    18
## 166  2.068084  2.28000    15
## 167  2.099657  2.54000    18
## 168  2.131231  2.46000    16
## 169  2.162805  2.54000    18
## 170  2.194379  2.28000    15
## 171  2.225953  2.54000    18
## 172  2.257526  2.28000    15
## 173  2.289100  2.28000    15
## 174  2.320674  2.53000    17
## 175  2.352248  2.53000    17
## 176  2.383822  2.46000    16
## 177  2.415395  2.28000    15
## 178  2.446969  2.53000    17
## 179  2.478543  2.54000    18
## 180  2.510117  2.54000    18
## 181  2.541691  2.54000    18
## 182  2.573264  2.54000    18
## 183  2.604838  2.54000    18
## 184  2.636412  2.46000    16
## 185  2.667986  2.53000    17
## 186  2.699560  2.99000    19
## 187  2.731133  2.99000    19
## 188  2.762707  2.99000    19
## 189  2.794281  2.54000    18
## 190  2.825855  2.54000    18
## 191  2.857428  2.28000    15
## 192  2.889002  2.53000    17
## 193  2.920576  2.53000    17
## 194  2.952150  2.53000    17
## 195  2.983724  2.99000    19
## 196  3.015297  2.99000    19
## 197  3.046871  3.88000    22
## 198  3.078445  3.70000    21
## 199  3.110019  3.91000    23
## 200  3.141593  3.53000    20
```