

华中师范大学

实验报告书

2022 年 12 月 18 日

课程名称:	时间序列分析
专 业:	统计学
年 级:	2020 级
学生姓名:	陈启源
学 号:	2020211946
指导教师:	张晓飞

华中师范大学数学与统计学学院

1 问题 1

1.1 问题重述

针对 AR(p) 模型，分别写出用矩估计、条件最小二乘估计的 R 程序，并验证你的程序。

1.2 问题分析

算法：

```
1 MM.AR <- function(Y, q){
2   n = length(Y)
3
4   r = my.acf(Y, q, plot = F)$acf
5   rr = c(1, r[seq(q-1)])
6
7   idx.mat = matrix(NA, q, q)
8   idx.mat[1,] = seq(0, q-1)
9   for(i in seq(2, q)){
10    idx.mat[i,] = idx.mat[i-1,]-1
11  }
12  idx.mat = abs(idx.mat)
13  idx.mat = idx.mat +1
14  A = matrix(NA, q, q)
15  for(i in seq(q)){
16    A[i,] = rr[idx.mat[i,]]
17  }
18
19  b = r[seq(q)]
20  phi = solve(A, b)
21
22  phi
23 }
24
25 my.acf = function(Y, lag.max=NULL, plot=TRUE, ci.type = "white"){
26
27   n = length(Y)
28   if(is.null(lag.max)){
29     lag.max = 20
30   }
```

```

31   if (max(lag.max)>=n){
32     stop("The largest lag.max need be smaller than n")
33   }
34
35   if (min(lag.max)<=0){
36     stop("The smallest lag.max need be large than 0")
37   }
38
39   Y.mean = Y - mean(Y)
40
41   acf.Y = rep(NA, lag.max)
42   for(k in seq(lag.max)){
43     acf.Y[k] = sum(Y.mean[1:(n-k)]*Y.mean[(k+1):n])
44   }
45   acf = acf.Y/sum(Y.mean*Y.mean)
46
47
48   #plot1.acf
49   if(plot){
50
51     ylim <- c(min(acf)-0.1, max(acf)+0.1)
52     plot(y = acf, x = 1:lag.max, type = "h",
53          xlab = "Lag", ylab = "ACF", ylim = ylim)
54     abline(h = 0, col="black")
55     if(ci.type == "ma"){
56       wt <- sqrt(cumsum(c(1, 2*acf^2)))
57       wt <- wt[-length(wt)]
58       lines(y = 1.96/sqrt(n)*wt, x = 1:lag.max, col="red", lty = 2)
59       lines(y = -1.96/sqrt(n)*wt, x = 1:lag.max, col="red", lty = 2)
60     }
61
62     if(ci.type == "white"){
63       abline(h = 1.96/sqrt(n), col="red", lty = 2)
64       abline(h = -1.96/sqrt(n), col="red", lty = 2)
65     }
66
67   }
68   out = list(acf=acf, Y= Y, lag.max=lag.max)

```

```

69 }
70
71 LS.AR <- function(Y, p){
72   n = length(Y)
73   mu = mean(Y)
74   Y.mean = Y - mu
75   X = matrix(0, n-p, p+1)
76   for (i in seq(n-p)){
77     X[i,] = c(1, Y.mean[seq(i+p-1, i, -1)])
78   }
79   y = Y.mean[(p+1):n]
80   phi = solve(t(X)%*%X, t(X)%*%y)
81   Intercept = phi[1]
82   phi = phi[-1]
83   result = list(mu=mu, phi=phi, Intercept=Intercept)
84 }

```

主程序：

```

1  library(TSA)
2  source("./ch7.R")
3  source("./sm_arma.R")
4
5  # 1
6  series = ARMA_func(c(0.1, 0.7), NULL, 100)
7  # 自己的结果
8  (MM.AR(series, 2))
9  (LS.AR(series, 2))
10 # 书本的结果
11 ar(series, order.max = 2, AIC = F, method = 'yw')
12 ar(series, order.max = 2, AIC = F, method = 'ols')
13
14
15 series = ARMA_func(c(0.2, 0.4, 0.6), NULL, 100)
16 (MM.AR(series, 3))
17 (LS.AR(series, 3))
18 ar(series, order.max = 3, AIC = F, method = 'yw')
19 ar(series, order.max = 3, AIC = F, method = 'ols')

```

结果:

参数	理论值	矩估计	条件最小二乘	书本矩估计	书本条件最小二乘
ϕ_1	0.1	0.1572	0.1339	0.1572	0.1339
ϕ_2	0.7	0.5704	0.6100	0.5705	0.6101

表 1: AR(2) 模型矩估计与条件最小二乘结果对比

参数	理论值	矩估计	条件最小二乘	书本矩估计	书本条件最小二乘
ϕ_1	0.2	0.9232	0.1138	0.9270	0.1138
ϕ_2	0.4	-0.0002	0.3216	0.0000	0.3216
ϕ_3	0.6	-0.0027	0.7856	0.0000	0.7856

表 2: AR(3) 模型矩估计与条件最小二乘结果对比

2 问题 2

2.1 问题重述

针对 AR(1) 模型，分别写出无条件最小二乘估计、极大似然估计的 R 程序，并验证你的程序。

2.2 问题分析

算法：

```
1 ULS.AR1 = function(data, intercept = T,
2                     tol = 1e-04,
3                     maxstep = 1e+04,
4                     LearningRate = 1){
5   if (intercept==T){
6     initial.ar = 0
7     initial.mu = 0
8     l = length(data)
9     G = 0
10    e = vector("numeric", length = l)
11    for (i in 2:l){
12      e[i] = data[i]-
13        initial.ar*data[i-1]+initial.mu*(initial.ar-1)
14    }
15    L.new = sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2
16    dEdAR = matrix(0, ncol = 1, nrow = 1)
17    dEdMU = matrix(0, ncol = 1, nrow = 1)
18    for (i in 2:l){
19      dEdAR[i] = initial.mu-data[i-1]
20      dEdMU[i] = initial.ar-1
21    }
22    dE = rbind(dEdAR, dEdMU)
23    de = rbind(-2*initial.ar*(data[1]-initial.mu)^2,
24              2*(1-initial.ar^2)*(initial.mu-data[1]))
25    d = 2*dE%*%e+de
26    G = G+sum(d^2)
27    delta = -d/sqrt(G)*LearningRate
28    p = 0
29    ar.new = initial.ar+delta[1]
30    mu.new = initial.mu+delta[2]
```

```

31 while (T){
32     e = vector("numeric", length = l)
33     for (i in 2:l){
34         e[i] = data[i]-ar.new*data[i-1]+mu.new*(ar.new-1)
35     }
36     L = L.new
37     L.new = sum(e^2)+(1-ar.new^2)*(data[1]-mu.new)^2
38     if (abs(L-L.new)<tol&sum(d^2)<tol){
39         para = c(ar1 = ar.new, intercept = mu.new)
40         print(as.symbol("Local minimum found."))
41         break
42     }
43     dEdAR = matrix(0, ncol = 1, nrow = 1)
44     dEdMU = matrix(0, ncol = 1, nrow = 1)
45     for (i in 2:l){
46         dEdAR[, i] = mu.new-data[i-1]
47         dEdMU[i] = ar.new-1
48     }
49     dE = rbind(dEdAR, dEdMU)
50     de = rbind(-2*ar.new*(data[1]-mu.new),
51               2*(1-ar.new^2)*(mu.new-data[1]))
52     d = 2*dE%%e+de
53     G = G+sum(d^2)
54     delta = -d/sqrt(G)*LearningRate
55     ar.new = ar.new+delta[1]
56     mu.new = mu.new+delta[2]
57     p = p+1
58     if (p>maxstep){
59         para = c(ar1 = ar.new, intercept = mu.new)
60         print(as.symbol("Number of iterations exceeded options."))
61         break
62     }
63 }
64 }else{
65     initial.ar = 0
66     l = length(data)
67     G = 0
68     e = vector("numeric", length = l)

```

```

69     for (i in 2:1){
70         e[i] = data[i]-initial.ar*data[i-1]
71     }
72     L.new = sum(e^2)+(1-initial.ar^2)*data[1]^2
73     dEdAR = matrix(0, ncol = 1, nrow = 1)
74     for (i in 2:1){
75         dEdAR[i] = -data[i-1]
76     }
77     de = -2*initial.ar*data[1]^2
78     d = 2*dEdAR%%e+de
79     G = G+d^2
80     delta = -d/sqrt(G)*LearningRate
81     p = 0
82     ar.new = initial.ar+delta
83     while (T){
84         e = vector("numeric", length = 1)
85         for (i in 2:1){
86             e[i] = data[i]-ar.new*data[i-1]
87         }
88         L = L.new
89         L.new = sum(e^2)+(1-ar.new^2)*data[1]^2
90         if (abs(L-L.new)<tol&sum(d^2)<tol){
91             para = c(ar1 = ar.new)
92             print(as.symbol("Local minimum found."))
93             break
94         }
95         dEdAR = matrix(0, ncol = 1, nrow = 1)
96         for (i in 2:1){
97             dEdAR[,i] = -data[i-1]
98         }
99         de = -2*ar.new*data[1]
100        d = 2*dEdAR%%e+de
101        G = G+d^2
102        delta = -d/sqrt(G)*LearningRate
103        ar.new = ar.new+delta
104        p = p+1
105        if (p>maxstep){
106            para = c(ar1 = ar.new)

```



```

107         print(as.symbol("Number of iterations exceeded options."))
108         break
109     }
110 }
111 }
112 return(para)
113 }
114
115
116
117 MLE.AR1 = function(data, intercept = T,
118                    tol = 1e-04,
119                    maxstep = 1e+04,
120                    LearningRate = 1){
121     #arima, order=c(1,0,0), method="ML"
122     if (intercept==T){
123         initial.ar = 0
124         initial.mu = 0
125         initial.sigma2 = 1
126         l = length(data)
127         G = 0
128         e = vector("numeric", length = l)
129         for (i in 2:l){
130             e[i] = data[i]-
131                 initial.ar*data[i-1]+initial.mu*(initial.ar-1)
132         }
133         L.new = -(sum(e^2)+
134                 (1-initial.ar^2)*(data[1]-initial.mu)^2)/2/initial.sigma2+
135                 -1/2*log(2*pi)-1/2*log(initial.sigma2)+1/2*log(1-initial.ar^2)
136         L.new = -L.new
137         dEdAR = matrix(0, ncol = 1, nrow = 1)
138         dEdMU = matrix(0, ncol = 1, nrow = 1)
139         for (i in 2:l){
140             dEdAR[i] = initial.mu-data[i-1]
141             dEdMU[i] = initial.ar-1
142         }
143         dLdAR = -initial.ar/(1-initial.ar^2)+
144             initial.ar*(data[1]-initial.mu)^2/initial.sigma2-

```

```

145     dEdAR%%e/initial.sigma2
146     dLdMU = (1-initial.ar^2)*(data[1]-initial.mu)/initial.sigma2-
147     dEdMU%%e/initial.sigma2
148     dLdS2 = -1/2/initial.sigma2+
149     (sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2)/2/initial.sigma2^2
150     dL = c(-dLdAR, -dLdMU, -dLdS2)
151     G = G+sum(dL^2)
152     delta = -dL/sqrt(G)*LearningRate
153     ar.new = initial.ar+delta[1]
154     mu.new = initial.mu+delta[2]
155     s2.new = initial.sigma2+delta[3]
156     p = 0
157     while (T){
158         e = vector("numeric", length = 1)
159         for (i in 2:l){
160             e[i] = data[i]-ar.new*data[i-1]+mu.new*(ar.new-1)
161         }
162         L = L.new
163         L.new = -(sum(e^2)+
164                 (1-ar.new^2)*(data[1]-mu.new)^2)/2/s2.new+
165                 -1/2*log(2*pi)-1/2*log(s2.new)+1/2*log(1-ar.new^2)
166         L.new = -L.new
167         if (abs(L-L.new)<tol&sum(dL^2)<tol){
168             para = c(ar1 = ar.new, intercept = mu.new)
169             print(as.symbol("Local minimum found. "))
170             break
171         }
172         dEdAR = matrix(0, ncol = 1, nrow = 1)
173         dEdMU = matrix(0, ncol = 1, nrow = 1)
174         for (i in 2:l){
175             dEdAR[, i] = mu.new-data[i-1]
176             dEdMU[i] = ar.new-1
177         }
178         dLdAR = -ar.new/(1-ar.new^2)+
179             ar.new*(data[1]-mu.new)^2/s2.new-
180             dEdAR%%e/s2.new
181         dLdMU = (1-ar.new^2)*(data[1]-mu.new)/s2.new-
182         dEdMU%%e/s2.new

```

```

183     dLdS2 = -1/2/s2.new+
184         (sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2)/2/s2.new^2
185     dL = c(-dLdAR, -dLdMU, -dLdS2)
186     G = G+sum(dL^2)
187     delta = -dL/sqrt(G)*LearningRate
188     ar.new = ar.new+delta[1]
189     mu.new = mu.new+delta[2]
190     s2.new = s2.new+delta[3]
191     p = p+1
192     if (p>maxstep){
193         para = c(ar1 = ar.new, intercept = mu.new)
194         print(as.symbol("Number of iterations exceeded options."))
195         break
196     }
197 }
198 }else{
199     initial.ar = 0
200     initial.sigma2 = 1
201     l = length(data)
202     G = 0
203     e = vector("numeric", length = l)
204     for (i in 2:l){
205         e[i] = data[i]-initial.ar*data[i-1]
206     }
207     L.new = -(sum(e^2)+
208         (1-initial.ar^2)*data[1]^2)/2/initial.sigma2+
209         -1/2*log(2*pi)-1/2*log(initial.sigma2)+1/2*log(1-initial.ar^2)
210     L.new = -L.new
211     dEdAR = matrix(0, ncol = 1, nrow = 1)
212     for (i in 2:l){
213         dEdAR[i] = -data[i-1]
214     }
215     dLdAR = -initial.ar/(1-initial.ar^2)+
216         initial.ar*data[1]^2/initial.sigma2-
217         dEdAR%%e/initial.sigma2
218     dLdS2 = -1/2/initial.sigma2+
219         (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/initial.sigma2^2
220     dL = c(-dLdAR, -dLdS2)

```

```

221 G = G+sum(dL^2)
222 delta = -dL/sqrt(G)*LearningRate
223 ar.new = initial.ar+delta[1]
224 s2.new = initial.sigma2+delta[2]
225 p = 0
226 while (T){
227     e = vector("numeric", length = 1)
228     for (i in 2:1){
229         e[i] = data[i]-ar.new*data[i-1]
230     }
231     L = L.new
232     L.new = -(sum(e^2)+
233              (1-ar.new^2)*data[1]^2)/2/s2.new+
234             -1/2*log(2*pi)-1/2*log(s2.new)+1/2*log(1-ar.new^2)
235     L.new = -L.new
236     if (abs(L-L.new)<tol&sum(dL^2)<tol){
237         para = c(ar1 = ar.new)
238         print(as.symbol("Local minimum found. "))
239         break
240     }
241     dEdAR = matrix(0, ncol = 1, nrow = 1)
242     for (i in 2:1){
243         dEdAR[,i] = -data[i-1]
244     }
245     dLdAR = -ar.new/(1-ar.new^2)+
246             ar.new*data[1]^2/s2.new-
247             dEdAR%%e/s2.new
248     dLdS2 = -1/2/s2.new+
249            (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/s2.new^2
250     dL = c(-dLdAR, -dLdS2)
251     G = G+sum(dL^2)
252     delta = -dL/sqrt(G)*LearningRate
253     ar.new = ar.new+delta[1]
254     s2.new = s2.new+delta[2]
255     p = p+1
256     if (p>maxstep){
257         para = c(ar1 = ar.new)
258         print(as.symbol("Number of iterations exceeded options. "))

```

```

259         break
260     }
261 }
262 }
263 return(para)
264 }

```

主程序:

```

1 # 2
2 data("ar1.s")
3 data("ar1.2.s")
4 source("../mlear1.R")
5 source("../ulsar1.R")
6 MLE.AR1(ar1.s)
7 ULS.AR1(ar1.s, LearningRate = 0.01)
8
9 MLE.AR1(ar1.2.s)
10 ULS.AR1(ar1.2.s, LearningRate = 0.01)

```

结果:

参数	书本极大似然	极大似然	书本无条件最小二乘	无条件最小二乘
0.9	0.892	0.8923	0.911	0.9146
0.4	0.465	0.4653	0.473	0.4734

表 3: AR(1) 程序结果与书本结果对比

3 问题 3

3.1 问题重述

针对 MA(q) 模型，写出用条件最小二乘估计的 R 程序，并验证你的程序。

3.2 问题分析

算法：

```
1 LS.MA <- function(Y, q, lr=0.1, maxit = 500){
2   n = length(Y)
3   N = n + q + 1
4   theta.old = rep(0, q)
5   loss = c()
6   for (i in seq(maxit)){
7     e = rep(0, N)
8     e.grad = matrix(0, q, N)
9     grad = rep(0, q)
10    for(j in seq(q+2, N)){
11      e[j] = Y[j-(q+1)] + sum(e[seq(j-1, j-q, -1)]*theta.old)
12      e.grad[, j] = e[seq(j-1, j-q, -1)] +
13      e.grad[, seq(j-1, j-q, -1)]%*%theta.old
14      grad = grad + 2*e[j]*e.grad[, j]
15    }
16    theta = theta.old - lr*grad
17    #print(grad)
18    loss[i] = sum(e*e)
19    #print(sum(abs(theta-theta.old)))
20    if (sum(abs(theta-theta.old)) < 1e-6)
21      break
22    theta.old = theta
23  }
24  result = list(theta=theta, loss=loss)
25  result
26 }
```

主程序：

```
1 # 3
2 set.seed(123)
```

```

3 series = ARMA_func(NULL, c(0.1, 0.7), 100)
4 (LS.MA(series, 2, lr=1e-4))
5 arima(series, order = c(0, 0, 2))$coef
6
7 series = ARMA_func(NULL, c(0.2, 0.4, 0.6), 100)
8 arima(series, order = c(0, 0, 3))$coef

```

结果:

参数	理论值	条件最小二乘	书本条件最小二乘
θ_1	0.1	0.0866	0.0877
θ_2	0.7	0.7142	0.7152

表 4: MA(2) 条件最小二乘参数估计

参数	理论值	条件最小二乘	书本条件最小二乘
θ_1	0.2	0.0543	0.0543
θ_2	0.4	0.2590	0.2590
θ_3	0.6	0.4798	0.4798

表 5: MA(3) 条件最小二乘参数估计

4 问题 4

4.1 问题重述

针对 ARMA(1,1) 模型，分别写出用矩估计、条件最小二乘估计的 R 程序，并验证你的程序。

4.2 问题分析

算法：

```
1 LS.ARMA11 = function(Y, lr=0.1, maxit = 500){
2   n = length(Y)
3   mu = mean(Y)
4   Y = Y - mu
5
6   theta.old = 0
7   phi.old = 0
8   loss = c()
9
10  grad.phi.total = 1
11  grad.phi.theta = 1
12
13  for (i in seq(maxit)){
14
15    e = rep(NA,n)
16    e.grad.phi = rep(NA, n)
17    e.grad.theta = rep(NA, n)
18
19    e[1] = Y[1]
20    e[2] = Y[2] - phi.old*Y[1] + theta.old*e[1]
21    e.grad.phi[1] = 0
22    e.grad.theta[1] = 0
23
24    e.grad.phi[2] = -Y[1]
25    e.grad.theta[2] = e[1]
26
27    for(j in seq(3,n)){
28      e[j] = Y[j] - phi.old*Y[j-1] + theta.old*e[j-1]
29      e.grad.phi[j] = -Y[j-1] + theta.old*e.grad.phi[j-1]
30      e.grad.theta[j] = e[j-1] + theta.old*e.grad.theta[j-1]
```



```

31     }
32
33     grad.phi = 2*mean(e*e.grad.phi)
34     grad.theta = 2*mean(e*e.grad.theta)
35
36     grad.phi.total = grad.phi.total + grad.phi^2
37     grad.phi.theta = grad.phi.theta + grad.theta^2
38
39     phi = phi.old - lr*grad.phi/sqrt(grad.phi.total)
40     theta = theta.old - lr*grad.theta/sqrt(grad.phi.theta)
41     loss[i] = mean(e*e)
42     if ((abs(phi-phi.old)+abs(theta-theta.old))<1e-5)
43         break
44     theta.old = theta
45     phi.old = phi
46 }
47 result = list(phi = phi, theta=theta, loss=loss)
48 result
49 }
50
51
52 MM.ARMA11 = function(data) {
53     result.acf = my.acf(data - mean(data), lag.max = 2, plot = FALSE)
54     acfval = result.acf$acf
55     phi = acfval [2] / acfval [1]
56     r1 = acfval [1]
57     a = r1 - phi
58     b = phi ^ 2 + 1 - 2 * r1 * phi
59     c = r1 - phi
60     x1 = (-b + sqrt(b ^ 2 - 4 * a * c)) / 2 / a
61     x2 = (-b - sqrt(b ^ 2 - 4 * a * c)) / 2 / a
62     theta = c(x1, x2)
63     theta = theta[which(abs(1 / theta) > 1)]
64     para = c(ar1 = phi, ma1 = theta)
65     return(para)
66 }

```

主程序：

```

1 # 4
2 # 书本结果
3 data("arma11.s")
4 MM.ARMA11(arma11.s)
5 LS.ARMA11(arma11.s, lr=0.1)
6 # 模拟数据
7 series = ARMA_func(c(0.7), c(0.5), 1000)
8 MM.ARMA11(series)
9 LS.ARMA11(series, lr = 0.1)
10
11 series = ARMA_func(c(0.3), c(0.6), 10000)
12 MM.ARMA11(series)
13 LS.ARMA11(series, lr = 0.1)

```

参数	理论值	矩估计	条件最小二乘	书本矩估计	书本条件最小二乘
ϕ	0.6	0.6378	0.5792	0.637	0.5586
θ	-0.3	-0.2038	-0.3246	-0.2066	-0.3669

表 6: ARMA(1,1) 过程书本计算结果与程序计算结果对比

通过以上的结果可以发现，程序计算的结果与理论值以及书本的参考结果误差较小，可以认为计算结果正确。

参数	理论值	矩估计	条件最小二乘
ϕ	0.7	0.6463	0.6216
θ	0.5	0.4082	0.3795

表 7: ARMA(1,1) 模拟过程 1

参数	理论值	矩估计	条件最小二乘
ϕ	0.3	0.2914	0.2975
θ	0.6	0.5999	0.6044

表 8: ARMA(1,1) 模拟过程 2