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：

```
MM.AR <- function(Y, q){
  n = length(Y)

  r = my.acf(Y, q, plot = F)$acf
  rr = c(1, r[seq(q-1)])

  idx.mat = matrix(NA,q,q)
  idx.mat[1,] =  seq(0,q-1)
  for(i in seq(2,q)){
    idx.mat[i,] = idx.mat[i-1,]-1
  }
  idx.mat = abs(idx.mat)
  idx.mat = idx.mat +1
  A = matrix(NA,q,q)
  for(i in seq(q)){
    A[i,] = rr[idx.mat[i,]]
  }

  b = r[seq(q)]
  phi = solve(A,b)

  phi
}

my.acf = function(Y, lag.max=NULL, plot=TRUE, ci.type = "white"){

  n = length(Y)
  if(is.null(lag.max)){
    lag.max = 20
  }
```

```r
  if(max(lag.max)>=n){
    stop("The largest lag.max need be smaller than n")
  }

  if(min(lag.max)<=0){
    stop("The smallest lag.max need be large than 0")
  }

  Y.mean = Y - mean(Y)

  acf.Y = rep(NA, lag.max)
  for(k in seq(lag.max)){
    acf.Y[k] = sum(Y.mean[1:(n-k)]*Y.mean[(k+1):n])
  }
  acf = acf.Y/sum(Y.mean*Y.mean)


  #plot1.acf
  if(plot){

    ylim <- c(min(acf)-0.1, max(acf)+0.1)
    plot(y = acf, x = 1:lag.max, type = "h",
    xlab = "Lag", ylab = "ACF", ylim = ylim)
    abline(h = 0, col="black")
    if(ci.type == "ma"){
      wt <- sqrt(cumsum(c(1, 2*acf^2)))
      wt <- wt[-length(wt)]
      lines(y = 1.96/sqrt(n)*wt, x = 1:lag.max, col="red", lty = 2)
      lines(y = -1.96/sqrt(n)*wt, x = 1:lag.max, col="red", lty = 2)
    }

    if(ci.type == "white"){
      abline(h = 1.96/sqrt(n), col="red", lty = 2)
      abline(h = -1.96/sqrt(n), col="red", lty = 2)
    }

  }
  out = list(acf=acf, Y= Y, lag.max=lag.max)
```

```
}

LS.AR <- function(Y, p){
  n = length(Y)
  mu = mean(Y)
  Y.mean = Y - mu
  X = matrix(0, n-p, p+1)
  for (i in seq(n-p)){
    X[i,] = c(1,Y.mean[seq(i+p-1,i,-1)])
  }
  y = Y.mean[(p+1):n]
  phi = solve(t(X)%*%X, t(X)%*%y)
  Intercept = phi[1]
  phi = phi[-1]
  result = list(mu=mu, phi=phi, Intercept=Intercept)
}
```

主程序：

```
library(TSA)
source("./ch7.R")
source("./sm_arma.R")

# 1
series = ARMA_func(c(0.1, 0.7), NULL, 100)
# 自己的结果
(MM.AR(series, 2))
(LS.AR(series, 2))
# 书本的结果
ar(series, order.max = 2, AIC = F, method = 'yw')
ar(series, order.max = 2, AIC = F, method = 'ols')


series = ARMA_func(c(0.2, 0.4, 06), NULL, 100)
(MM.AR(series, 3))
(LS.AR(series, 3))
ar(series, order.max = 3, AIC = F, method = 'yw')
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：

```
ULS.AR1 = function(data, intercept = T,
                          tol = 1e-04,
                          maxstep = 1e+04,
                          LearningRate = 1){
  if (intercept==T){
     initial.ar = 0
     initial.mu = 0
     l = length(data)
     G = 0
     e = vector("numeric", length = l)
     for (i in 2:l){
        e[i] = data[i]-
           initial.ar*data[i-1]+initial.mu*(initial.ar-1)
     }
     L.new = sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2
     dEdAR = matrix(0, ncol = l, nrow = 1)
     dEdMU = matrix(0, ncol = l, nrow = 1)
     for (i in 2:l){
        dEdAR[i] = initial.mu-data[i-1]
        dEdMU[i] = initial.ar-1
     }
     dE = rbind(dEdAR, dEdMU)
     de = rbind(-2*initial.ar*(data[1]-initial.mu)^2,
                 2*(1-initial.ar^2)*(initial.mu-data[1]))
     d = 2*dE%*%e+de
     G = G+sum(d^2)
     delta = -d/sqrt(G)*LearningRate
     p = 0
     ar.new = initial.ar+delta[1]
     mu.new = initial.mu+delta[2]
```

```
    while (T){
      e = vector("numeric", length = l)
      for (i in 2:l){
        e[i] = data[i]-ar.new*data[i-1]+mu.new*(ar.new-1)
      }
      L = L.new
      L.new = sum(e^2)+(1-ar.new^2)*(data[1]-mu.new)^2
      if (abs(L-L.new)<tol&sum(d^2)<tol){
        para = c(ar1 = ar.new, intercept = mu.new)
        print(as.symbol("Local minimum found."))
        break
      }
      dEdAR = matrix(0, ncol = l, nrow = 1)
      dEdMU = matrix(0, ncol = l, nrow = 1)
      for (i in 2:l){
        dEdAR[,i] = mu.new-data[i-1]
        dEdMU[i] = ar.new-1
      }
      dE = rbind(dEdAR, dEdMU)
      de = rbind(-2*ar.new*(data[1]-mu.new),
                 2*(1-ar.new^2)*(mu.new-data[1]))
      d = 2*dE%*%e+de
      G = G+sum(d^2)
      delta = -d/sqrt(G)*LearningRate
      ar.new = ar.new+delta[1]
      mu.new = mu.new+delta[2]
      p = p+1
      if (p>maxstep){
        para = c(ar1 = ar.new, intercept = mu.new)
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }else{
    initial.ar = 0
    l = length(data)
    G = 0
    e = vector("numeric", length = l)
```

```
    for (i in 2:l){
      e[i] = data[i]-initial.ar*data[i-1]
    }
    L.new = sum(e^2)+(1-initial.ar^2)*data[1]^2
    dEdAR = matrix(0, ncol = l, nrow = 1)
    for (i in 2:l){
      dEdAR[i] = -data[i-1]
    }
    de = -2*initial.ar*data[1]^2
    d = 2*dEdAR%*%e+de
    G = G+d^2
    delta = -d/sqrt(G)*LearningRate
    p = 0
    ar.new = initial.ar+delta
    while (T){
      e = vector("numeric", length = l)
      for (i in 2:l){
        e[i] = data[i]-ar.new*data[i-1]
      }
      L = L.new
      L.new = sum(e^2)+(1-ar.new^2)*data[1]^2
      if (abs(L-L.new)<tol&sum(d^2)<tol){
        para = c(ar1 = ar.new)
        print(as.symbol("Local minimum found."))
        break
      }
      dEdAR = matrix(0, ncol = l, nrow = 1)
      for (i in 2:l){
        dEdAR[,i] = -data[i-1]
      }
      de = -2*ar.new*data[1]
      d = 2*dEdAR%*%e+de
      G = G+d^2
      delta = -d/sqrt(G)*LearningRate
      ar.new = ar.new+delta
      p = p+1
      if (p>maxstep){
        para = c(ar1 = ar.new)
```

```
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }
  return(para)
}



MLE.AR1 = function(data, intercept = T,
                        tol = 1e-04,
                        maxstep = 1e+04,
                        LearningRate = 1){
  #arima,order=c(1,0,0),method="ML"
  if (intercept==T){
    initial.ar = 0
    initial.mu = 0
    initial.sigma2 = 1
    l = length(data)
    G = 0
    e = vector("numeric", length = l)
    for (i in 2:l){
      e[i] = data[i]-
        initial.ar*data[i-1]+initial.mu*(initial.ar-1)
    }
    L.new = -(sum(e^2)+
              (1-initial.ar^2)*(data[1]-initial.mu)^2)/2/initial.sigma2+
      -l/2*log(2*pi)-l/2*log(initial.sigma2)+1/2*log(1-initial.ar^2)
    L.new = -L.new
    dEdAR = matrix(0, ncol = l, nrow = 1)
    dEdMU = matrix(0, ncol = l, nrow = 1)
    for (i in 2:l){
      dEdAR[i] = initial.mu-data[i-1]
      dEdMU[i] = initial.ar-1
    }
    dLdAR = -initial.ar/(1-initial.ar^2)+
      initial.ar*(data[1]-initial.mu)^2/initial.sigma2-
```

```
    dEdAR%*%e/initial.sigma2
  dLdMU = (1-initial.ar^2)*(data[1]-initial.mu)/initial.sigma2-
    dEdMU%*%e/initial.sigma2
  dLdS2 = -1/2/initial.sigma2+
    (sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2)/2/initial.sigma2^2
  dL = c(-dLdAR, -dLdMU, -dLdS2)
  G = G+sum(dL^2)
  delta = -dL/sqrt(G)*LearningRate
  ar.new = initial.ar+delta[1]
  mu.new = initial.mu+delta[2]
  s2.new = initial.sigma2+delta[3]
  p = 0
  while (T){
    e = vector("numeric", length = l)
    for (i in 2:l){
      e[i] = data[i]-ar.new*data[i-1]+mu.new*(ar.new-1)
    }
    L = L.new
    L.new = -(sum(e^2)+
                  (1-ar.new^2)*(data[1]-mu.new)^2)/2/s2.new+
      -l/2*log(2*pi)-l/2*log(s2.new)+1/2*log(1-ar.new^2)
    L.new = -L.new
    if (abs(L-L.new)<tol&sum(dL^2)<tol){
      para = c(ar1 = ar.new, intercept = mu.new)
      print(as.symbol("Local minimum found."))
      break
    }
    dEdAR = matrix(0, ncol = l, nrow = 1)
    dEdMU = matrix(0, ncol = l, nrow = 1)
    for (i in 2:l){
      dEdAR[,i] = mu.new-data[i-1]
      dEdMU[i] = ar.new-1
    }
    dLdAR = -ar.new/(1-ar.new^2)+
      ar.new*(data[1]-mu.new)^2/s2.new-
      dEdAR%*%e/s2.new
    dLdMU = (1-ar.new^2)*(data[1]-mu.new)/s2.new-
      dEdMU%*%e/s2.new
```

```
        dLdS2 = -1/2/s2.new+
          (sum(e^2)+(1-initial.ar^2)*(data[1]-initial.mu)^2)/2/s2.new^2
        dL = c(-dLdAR, -dLdMU, -dLdS2)
        G = G+sum(dL^2)
        delta = -dL/sqrt(G)*LearningRate
        ar.new = ar.new+delta[1]
        mu.new = mu.new+delta[2]
        s2.new = s2.new+delta[3]
        p = p+1
        if (p>maxstep){
          para = c(ar1 = ar.new, intercept = mu.new)
          print(as.symbol("Number of iterations exceeded options."))
          break
        }
      }
  }else{
      initial.ar = 0
      initial.sigma2 = 1
      l = length(data)
      G = 0
      e = vector("numeric", length = l)
      for (i in 2:l){
        e[i] = data[i]-initial.ar*data[i-1]
      }
      L.new = -(sum(e^2)+
                  (1-initial.ar^2)*data[1]^2)/2/initial.sigma2+
        -l/2*log(2*pi)-l/2*log(initial.sigma2)+1/2*log(1-initial.ar^2)
      L.new = -L.new
      dEdAR = matrix(0, ncol = l, nrow = 1)
      for (i in 2:l){
        dEdAR[i] = -data[i-1]
      }
      dLdAR = -initial.ar/(1-initial.ar^2)+
        initial.ar*data[1]^2/initial.sigma2-
        dEdAR%*%e/initial.sigma2
      dLdS2 = -l/2/initial.sigma2+
        (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/initial.sigma2^2
      dL = c(-dLdAR, -dLdS2)
```

```
  G = G+sum(dL^2)
  delta = -dL/sqrt(G)*LearningRate
  ar.new = initial.ar+delta[1]
  s2.new = initial.sigma2+delta[2]
  p = 0
  while (T){
    e = vector("numeric", length = l)
    for (i in 2:l){
      e[i] = data[i]-ar.new*data[i-1]
    }
    L = L.new
    L.new = -(sum(e^2)+
                  (1-ar.new^2)*data[1]^2)/2/s2.new+
      -l/2*log(2*pi)-l/2*log(s2.new)+1/2*log(1-ar.new^2)
    L.new = -L.new
    if (abs(L-L.new)<tol&sum(dL^2)<tol){
      para = c(ar1 = ar.new)
      print(as.symbol("Local minimum found."))
      break
    }
    dEdAR = matrix(0, ncol = l, nrow = 1)
    for (i in 2:l){
      dEdAR[,i] = -data[i-1]
    }
    dLdAR = -ar.new/(1-ar.new^2)+
      ar.new*data[1]^2/s2.new-
      dEdAR%*%e/s2.new
    dLdS2 = -l/2/s2.new+
      (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/s2.new^2
    dL = c(-dLdAR, -dLdS2)
    G = G+sum(dL^2)
    delta = -dL/sqrt(G)*LearningRate
    ar.new = ar.new+delta[1]
    s2.new = s2.new+delta[2]
    p = p+1
    if (p>maxstep){
      para = c(ar1 = ar.new)
      print(as.symbol("Number of iterations exceeded options."))
```

```
            break
          }
        }
    }
    return(para)
}
```

主程序：

```
# 2
data("ar1.s")
data("ar1.2.s")
source("./mlear1.R")
source("./ulsar1.R")
MLE.AR1(ar1.s)
ULS.AR1(ar1.s,LearningRate = 0.01)

MLE.AR1(ar1.2.s)
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：

```
LS.MA <- function(Y, q, lr=0.1, maxit = 500){
  n = length(Y)
  N = n + q +1
  theta.old = rep(0, q)
  loss = c()
  for (i in seq(maxit)){
    e = rep(0,N)
    e.grad = matrix(0, q, N)
    grad = rep(0,q)
    for(j in seq(q+2,N)){
      e[j] = Y[j-(q+1)] + sum(e[seq(j-1,j-q,-1)]*theta.old)
      e.grad[,j] = e[seq(j-1,j-q,-1)] +
      e.grad[,seq(j-1,j-q,-1)]%*%theta.old
      grad = grad + 2*e[j]*e.grad[,j]
    }
    theta = theta.old - lr*grad
    #print(grad)
    loss[i] = sum(e*e)
    #print(sum(abs(theta-theta.old)))
    if (sum(abs(theta-theta.old)) < 1e-6)
      break
    theta.old = theta
  }
  result = list(theta=theta, loss=loss)
  result
}
```

主程序：

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

```r
series = ARMA_func(NULL, c(0.1, 0.7), 100)
(LS.MA(series,2,lr=1e-4))
arima(series,order = c(0,0,2))$coef

series = ARMA_func(NULL, c(0.2, 0.4, 0.6), 100)
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:

```
LS.ARMA11 =  function(Y, lr=0.1, maxit = 500){
  n = length(Y)
  mu = mean(Y)
  Y = Y - mu

  theta.old = 0
  phi.old = 0
  loss = c()

  grad.phi.total =1
  grad.phi.theta =1

  for (i in seq(maxit)){

    e = rep(NA,n)
    e.grad.phi = rep(NA, n)
    e.grad.theta = rep(NA, n)

    e[1] = Y[1]
    e[2] = Y[2] - phi.old*Y[1] + theta.old*e[1]
    e.grad.phi[1] = 0
    e.grad.theta[1] = 0

    e.grad.phi[2] = -Y[1]
    e.grad.theta[2] = e[1]

    for(j in seq(3,n)){
      e[j] = Y[j] - phi.old*Y[j-1] + theta.old*e[j-1]
      e.grad.phi[j] = -Y[j-1] + theta.old*e.grad.phi[j-1]
      e.grad.theta[j] = e[j-1] + theta.old*e.grad.theta[j-1]
```

```
    }

    grad.phi = 2*mean(e*e.grad.phi)
    grad.theta = 2*mean(e*e.grad.theta)

    grad.phi.total = grad.phi.total + grad.phi^2
    grad.phi.theta = grad.phi.theta + grad.theta^2

    phi = phi.old - lr*grad.phi/sqrt(grad.phi.total)
    theta = theta.old - lr*grad.theta/sqrt(grad.phi.theta)
    loss[i] = mean(e*e)
    if ((abs(phi-phi.old)+abs(theta-theta.old))<1e-5)
      break
    theta.old = theta
    phi.old = phi
  }
  result = list(phi = phi, theta=theta, loss=loss)
  result
}


MM.ARMA11 = function(data) {
  result.acf = my.acf(data - mean(data), lag.max = 2, plot = FALSE)
  acfval = result.acf$acf
  phi = acfval [2] / acfval [1]
  r1 = acfval [1]
  a = r1 - phi
  b = phi ^ 2 + 1 - 2 * r1 * phi
  c = r1 - phi
  x1 = (-b + sqrt(b ^ 2 - 4 * a * c)) / 2 / a
  x2 = (-b - sqrt(b ^ 2 - 4 * a * c)) / 2 / a
  theta = c(x1, x2)
  theta = theta[which(abs(1 / theta) > 1)]
  para = c(ar1 = phi, ma1 = theta)
  return(para)
}
```

主程序：

```
# 4
# 书本结果
data("arma11.s")
MM.ARMA11(arma11.s)
LS.ARMA11(arma11.s,lr=0.1)
# 模拟数据
series = ARMA_func(c(0.7), c(0.5), 1000)
MM.ARMA11(series)
LS.ARMA11(series, lr = 0.1)

series = ARMA_func(c(0.3), c(0.6), 10000)
MM.ARMA11(series)
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