

# 时间序列分析作业与第七次实验报告

姓名：康江睿

学号：2018213779

指导老师： 张晓飞

2020 年 12 月 29 日

## 1 矩估计

### 1.1 对AR(p)模型参数的矩估计

对于一般的AR(p)模型，其矩估计的推导过程如下：

如果用数据的样本自相关函数 $\{r_1, r_2, \dots, r_p\}$ 代替生成数据的AR(p)模型的理论自相关函数 $\{\rho_1, \rho_2, \dots, \rho_p\}$ ，那么Yule-Walker方程变为：

$$\begin{cases} \phi_1 + r_1\phi_2 + \dots + r_{p-1}\phi_p = r_1 \\ r_1\phi_1 + \phi_2 + \dots + r_{p-2}\phi_p = r_2 \\ \dots \\ r_{p-1}\phi_1 + r_{p-2}\phi_2 + \dots + \phi_p = r_p \end{cases}$$

解这个方程就可以解得 $\{\phi_1, \phi_2, \dots, \phi_p\}$ 的矩估计。

### 1.2 对ARMA(1,1)模型参数的矩估计

回忆：ARMA(1,1)模型的理论自相关函数满足：

$$\rho_k = \frac{(1 - \theta\phi)(\phi - \theta)}{1 - 2\theta\phi + \theta^2} \phi^{k-1}, \quad k \geq 1$$

而 $\rho_2/\rho_1 = \phi$ 。那么以样本自相关函数代替理论自相关函数，得到：

$$\hat{\phi} = \frac{r_2}{r_1}$$

那么就可以进一步得到：

$$r_1 = \frac{(1 - \theta\hat{\phi})(\hat{\phi} - \theta)}{1 - 2\theta\hat{\phi} + \theta^2}$$

求解这个一元二次方程可以得到 $\hat{\theta}$ ，但是需要注意保留可逆解。

## 2 数值算法

### 2.1 梯度下降法(Gradient Descent)

对于一个无约束最优化问题

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x})$$

其中 $f(\mathbf{x})$ 是 $\mathbf{R}^n$ 上具有一阶连续偏导的函数。那么如果选取适当的初值 $\mathbf{x}^{(0)}$ ，我们可以通过梯度下降法来求得这个函数的某个局部最小值。算法过程如下：

---

**Algorithm 1** 梯度下降法 (Gradient Descent)

---

**Require:** 目标函数 $f(\mathbf{x}) \in \mathbf{R}^n$ ; 梯度函数 $g(\mathbf{x}) = \nabla f(\mathbf{x})$ ; 初始迭代值 $\mathbf{x}^{(0)} \in \mathbf{R}^n$ ; 计算精度 $\epsilon$ ; 学习率 $\lambda$

**Ensure:**  $f(\mathbf{x})$ 的某个局部极小值点 $\mathbf{x}^*$

- 1: 取初始值 $\mathbf{x}^{(0)}$ , 置 $k = 0$ ;
  - 2: **repeat**
  - 3:   计算 $f(\mathbf{x}^{(k)})$ 和 $g_k = g(\mathbf{x}^{(k)}) = \nabla f(\mathbf{x}^{(k)})$ ;
  - 4:   置 $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \lambda g_k$ ;
  - 5:   计算 $f(\mathbf{x}^{(k+1)})$ 和 $g_{k+1} = g(\mathbf{x}^{(k+1)}) = \nabla f(\mathbf{x}^{(k+1)})$ ;
  - 6:   置 $k = k + 1$
  - 7: **until** ( $\|g_k\| < \epsilon$ )或者( $|f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k-1)})| < \epsilon$ );
  - 8: **return**  $\mathbf{x}^* = \mathbf{x}^{(k)}$
- 

当目标函数是严格的凸函数时，梯度下降法的解保证为全局最优解；一般情况下则只能保证为局部最优解。此外，梯度下降法的收敛速度也未必很快。如果学习率定的太大，则在接近收敛的时候可能会发生振荡的现象；如果学习率定的太小，则算法收敛速度会很慢，迭代次数会非常大。为了解决这个问题，我们引入Adagrad算法。

### 2.2 自适应梯度下降法(Adaptive Gradient Descent)

定义在梯度下降法中

$$\Delta \mathbf{x}_k = -\lambda g_k = -\lambda \nabla f(\mathbf{x}^{(k)})$$

自适应梯度下降法中则是这样一种情况

$$\begin{aligned} n_k &= n_{k-1} + \|g_k\|_2^2, \quad n_0 = 0 \\ \Delta \mathbf{x}_k &= -\frac{\lambda}{\sqrt{n_k + \delta}} g_k, \quad \delta > 0 \end{aligned}$$

我们可以发现，自适应梯度下降法采取自适应调整学习率的方法，前期激励收敛，后期惩罚收敛，使得学习率随着迭代次数增加而递减，从而不会发生振荡的问题。但是我们需要注意，学习率需要设定得比较大，否则自适应学习率会提前收敛，从而导致算法收敛速度提前减慢，而无法得到结果。

梯度下降法中最重要的就是梯度的计算。以下对各个目标函数的梯度进行推导。

### 2.3 条件最小二乘：ARMA(p,q)模型

对于ARMA(p,q)过程：

$$Y_t = \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} +$$

$$e_t - \theta_1 e_{t-1} - \theta_2 e_{t-2} - \cdots - \theta_q e_{t-q}$$

$$\text{记 } \boldsymbol{\phi} = [\phi_1, \phi_2, \cdots, \phi_p], \boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_q]$$

于是记

$$e_t(\boldsymbol{\phi}, \boldsymbol{\theta}) = e_t = Y_t - \phi_1 Y_{t-1} - \cdots - \phi_p Y_{t-p} + \theta_1 e_{t-1} + \cdots + \theta_q e_{t-q}$$

由最小二乘法的思想，我们将  $S_c(\boldsymbol{\phi}, \boldsymbol{\theta}) = \sum_{t=p+1}^n e_t^2$  作为最小化的目标函数。如果令  $e_0 = e_{-1} = \cdots = e_{-q+1} = 0$  以及  $Y_0 = Y_{-1} = \cdots = Y_{-p+1}$ ，我们称这个方法为条件最小二乘。而  $S_c(\boldsymbol{\phi}, \boldsymbol{\theta})$  的偏导数可以转化为如下结果（以自变量为  $\theta_1$  为例）：

$$\frac{\partial}{\partial \theta_1} S_c(\boldsymbol{\phi}, \boldsymbol{\theta}) = 2 \sum_{t=p+1}^n e_t \frac{\partial}{\partial \theta_1} e_t$$

因此只要求得  $e_t$  和  $e_t$  对所有参数的导数，我们就能计算  $S_c(\boldsymbol{\phi}, \boldsymbol{\theta})$  对所有参数的导数。首先， $e_t$  可以这样递归地获得：

$$\begin{cases} e_1 = Y_1 - \phi_1 Y_0 - \cdots - \phi_p Y_{1-p} + \theta_1 e_0 + \cdots + \theta_q e_{1-q} \\ e_2 = Y_2 - \phi_1 Y_1 - \cdots - \phi_p Y_{2-p} + \theta_1 e_0 + \cdots + \theta_q e_{2-q} \\ \cdots \\ e_t = Y_t - \phi_1 Y_{t-1} - \cdots - \phi_p Y_{t-p} + \theta_1 e_{t-1} + \cdots + \theta_q e_{t-q} \end{cases}$$

再由  $e_t$  的表达式我们计算它对AR类参数  $\{\phi_i, i = 1, \cdots, p\}$  和MA类参数  $\{\theta_j, j = 1, \cdots, q\}$  的偏导数：

$$\begin{cases} \frac{\partial}{\partial \phi_i} e_t = -Y_{t-i} + \theta_1 \frac{\partial}{\partial \phi_i} e_{t-1} + \cdots + \theta_q \frac{\partial}{\partial \phi_i} e_{t-q}, i = 1, \cdots, p \\ \frac{\partial}{\partial \theta_j} e_t = e_{t-j} + \theta_1 \frac{\partial}{\partial \theta_j} e_{t-1} + \cdots + \theta_q \frac{\partial}{\partial \theta_j} e_{t-q}, j = 1, \cdots, q \end{cases}$$

我们仍然可以递归地计算出这些偏导数。这样下来，我们就可以得到  $S_c(\boldsymbol{\phi}, \boldsymbol{\theta})$  的所有偏导数。

当模型中存在均值参数时，即：

$$Y_t - \mu = \phi_1(Y_{t-1} - \mu) + \cdots + \phi_p(Y_{t-p} - \mu) +$$

$$e_t - \theta_1 e_{t-1} - \cdots - \theta_q e_{t-q}$$

$$i.e. \quad Y_t = \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} +$$

$$e_t - \theta_1 e_{t-1} - \cdots - \theta_q e_{t-q} + (1 - \sum_p \phi_i) \mu$$

$e_t$ 的表达式变为

$$e_t(\phi, \theta, \mu) = Y_t - \phi_1 Y_{t-1} - \cdots - \phi_p Y_{t-p} +$$

$$\theta_1 e_{t-1} + \cdots + \theta_q e_{t-q} - (1 - \sum_p \phi_i) \mu$$

最小化的目标函数变为  $S_c(\phi, \theta, \mu) = \sum_{t=p+1}^n e_t^2$ ，其偏导数的形式不变。 $e_t$ 的计算思路不变，只是递推式要变为 $e_t$ 的新表达式。 $e_t$ 的偏导数的计算如下：

$$\begin{cases} \frac{\partial}{\partial \phi_i} e_t = -Y_{t-i} + \theta_1 \frac{\partial}{\partial \phi_i} e_{t-1} + \cdots + \theta_q \frac{\partial}{\partial \phi_i} e_{t-q}, & i = 1, \cdots, p \\ \frac{\partial}{\partial \theta_j} e_t = e_{t-j} + \theta_1 \frac{\partial}{\partial \theta_j} e_{t-1} + \cdots + \theta_q \frac{\partial}{\partial \theta_j} e_{t-q}, & j = 1, \cdots, q \\ \frac{\partial}{\partial \mu} e_t = (\sum_p \phi_i - 1) + \theta_1 \frac{\partial}{\partial \mu} e_{t-1} + \cdots + \theta_q \frac{\partial}{\partial \mu} e_{t-q} \end{cases}$$

于是，我们可以得到 $S_c(\phi, \theta, \mu)$ 的所有偏导数。

## 2.4 极大似然估计：AR(1)模型

对于AR(1)过程

$$Y_t = \phi Y_{t-1} + e_t$$

观测样本 $Y_1, Y_2, \cdots, Y_t$ 的对数似然函数如下：

$$l(\phi, \sigma_e^2) = -\frac{n \log(2\pi)}{2} - \frac{n \log(\sigma_e^2)}{2} + \frac{\log(1 - \phi^2)}{2} - \frac{S(\phi)}{2\sigma_e^2}$$

其中，

$$\begin{aligned} S(\phi) &= \sum_{t=2}^n e_t^2 + (1 - \phi^2) Y_1^2 \\ &= \sum_{t=2}^n (Y_t - \phi Y_{t-1})^2 + (1 - \phi^2) Y_1^2 \end{aligned}$$

$l$ 关于两个参数的偏导计算如下：

$$\begin{cases} \frac{\partial}{\partial \phi} l = -\frac{\phi}{1 - \phi^2} + \frac{1}{2\sigma_e^2} \sum_{t=2}^n e_t \frac{\partial}{\partial \phi} e_t + \frac{\phi Y_1^2}{\sigma_e^2} \\ \frac{\partial}{\partial \sigma_e^2} l = -\frac{n}{2\sigma_e^2} + \frac{1}{2\sigma_e^4} \sum_{t=2}^n e_t^2 \end{cases}$$

当模型中存在均值参数时，即：

$$Y_t - \mu = \phi(Y_{t-1} - \mu) + e_t$$

$$i.e. \quad Y_t = \phi Y_{t-1} + e_t + (1 - \phi) \mu$$

此时观测样本 $Y_1, Y_2, \dots, Y_t$ 的对数似然函数如下:

$$l(\phi, \sigma_e^2) = -\frac{n \log(2\pi)}{2} - \frac{n \log(\sigma_e^2)}{2} + \frac{\log(1 - \phi^2)}{2} - \frac{S(\phi, \mu)}{2\sigma_e^2}$$

其中,

$$\begin{aligned} S(\phi, \mu) &= \sum_{t=2}^n e_t^2 + (1 - \phi^2)(Y_1 - \mu)^2 \\ &= \sum_{t=2}^n [(Y_t - \mu) - \phi(Y_{t-1} - \mu)]^2 + (1 - \phi^2)(Y_1 - \mu)^2 \end{aligned}$$

$l$ 关于三个参数的偏导计算如下:

$$\begin{cases} \frac{\partial}{\partial \phi} l = -\frac{\phi}{1-\phi^2} + \frac{1}{2\sigma_e^2} \sum_{t=2}^n e_t \frac{\partial}{\partial \phi} e_t + \frac{\phi(Y_1 - \mu)^2}{\sigma_e^2} \\ \frac{\partial}{\partial \mu} l = -\frac{1}{2\sigma_e^2} \sum_{t=2}^n e_t \frac{\partial}{\partial \mu} e_t + \frac{(1-\phi^2)(Y_1 - \mu)}{\sigma_e^2} \\ \frac{\partial}{\partial \sigma_e^2} l = -\frac{n}{2\sigma_e^2} + \frac{1}{2\sigma_e^4} \sum_{t=2}^n e_t^2 \end{cases}$$

于是, 我们可以得到 $l(\phi, \mu, \sigma_e^2)$ 的所有偏导数。注意到我们要极大化 $l$ , 因此应该最小化负对数似然函数。

## 2.5 无条件最小二乘: AR(1)模型

在CSS和ML中做个折中, 我们以最小化 $S(\phi, \mu)$ 作为优化目标。其偏导数计算其实已经包括在了上一节的内容中, 故不多阐述。

# 3 R语言实现

## 3.1 矩估计

矩估计的函数要求使用函数`acfun` (第六章作业中的自定义函数), 故需先导入该函数

```
1 source('D:/R Files/TSACourse/acfun.R', encoding = 'UTF-8')
```

以下为对AR(p)模型参数进行矩估计的函数

```
1 me.ar = function(data, order){
2   #ar.yw
3   result.acf = acfun(data = data - mean(data), lag.max = order,
4                       plot = FALSE)
5   acfval = result.acf$acfval
6   phi = integer(order)
7   for (i in 1:order){
8     if (i==1){
9       RHO = 1
10    }else{
```

```

11     RHO = rbind(c(1, acfval[1:(i-1)]), cbind(acfval[1:(i-1)], RHO))
12   }
13 }
14 phi = as.numeric(qr.solve(RHO, acfval))
15 name = NULL
16 for (i in 1:order){
17   name[i] = paste("ar", as.character(i), sep = "")
18 }
19 names(phi) = name
20 para = phi
21 return(para)
22 }

```

以下为对ARMA(1,1)模型参数进行矩估计的函数

```

1 me.arma11 = function(data){
2   result.acf = acfun(data = data - mean(data), lag.max = 2, plot = FALSE)
3   acfval = result.acf$acfval
4   phi = acfval[2] / acfval[1]
5   r1 = acfval[1]
6   a = r1 - phi; b = phi^2 + 1 - 2*r1*phi; c = r1 - phi
7   x1 = (-b + sqrt(b^2 - 4*a*c)) / 2 / a
8   x2 = (-b - sqrt(b^2 - 4*a*c)) / 2 / a
9   theta = c(x1, x2)
10  theta = theta[which(abs(1/theta) > 1)]
11  para = c(ar1 = phi, ma1 = theta)
12  return(para)
13 }

```

## 3.2 条件最小二乘

以下为对ARMA(p,q)模型参数进行条件最小二乘估计的函数

```

1 clse.arma = function(data, order, intercept = T,
2   tol = 1e-04,
3   maxstep = 1e+04,
4   LearningRate = 0.1){
5   #arima,method="CSS"
6   if (intercept == T){
7     order.ar = order[1]
8     order.ma = order[2]
9     initial.ar = as.numeric(integer(order[1]))
10    initial.ma = as.numeric(integer(order[2]))
11    initial.mu = 0
12    l = length(data)
13    G = 0
14    if (order.ma == 0){

```

```

15     e = vector("numeric", length = 1)
16     for (i in (order.ar+1):l){
17         e[i] = data[i]-
18             initial.ar%%rev(data[(i-order.ar):(i-1)])+
19             initial.mu*(sum(initial.ar)-1)
20     }
21     L.new = sum(e^2)
22     dEdAR = matrix(0, ncol = 1, nrow = order.ar)
23     dEdMU = matrix(0, ncol = 1, nrow = 1)
24     for (i in (order.ar+1):l){
25         dEdAR[,i] = rep(initial.mu, order.ar)-
26             rev(data[(i-order.ar):(i-1)])
27         dEdMU[i] = sum(initial.ar)-1
28     }
29     dE = rbind(dEdAR, dEdMU)
30     d = 2*dE%%e
31     G = G+sum(d^2)
32     delta = -d/sqrt(G)*LearningRate
33     p = 0
34     ar.new = initial.ar+delta[1:order.ar]
35     mu.new = initial.mu+delta[1+order.ar]
36     while (T){
37         e = vector("numeric", length = 1)
38         for (i in (order.ar+1):l){
39             e[i] = data[i]-
40                 ar.new%%rev(data[(i-order.ar):(i-1)])+
41                 mu.new*(sum(ar.new)-1)
42         }
43         L = L.new
44         L.new = sum(e^2)
45         if (abs(L-L.new)<tol&sum(d^2)<tol){
46             name = NULL
47             for (i in 1:order.ar){
48                 name[i] = paste("ar", as.character(i), sep = "")
49             }
50             names(ar.new) = name
51             para = c(ar.new, intercept = mu.new)
52             print(as.symbol("Local minimum found. "))
53             break
54         }
55         dEdAR = matrix(0, ncol = 1, nrow = order.ar)
56         dEdMU = matrix(0, ncol = 1, nrow = 1)
57         for (i in (order.ar+1):l){
58             dEdAR[,i] =
59                 rep(mu.new, order.ar)-rev(data[(i-order.ar):(i-1)])
60             dEdMU[i] = sum(ar.new)-1
61         }
62         dE = rbind(dEdAR, dEdMU)

```

```

63     d = 2*dE%%e
64     G = G+sum(d^2)
65     delta = -d/sqrt(G)*LearningRate
66     ar.new = ar.new+delta[1:order.ar]
67     mu.new = mu.new+delta[1+order.ar]
68     p = p+1
69     if (p>maxstep){
70         name = NULL
71         for (i in 1:order.ar){
72             name[i] = paste("ar", as.character(i), sep = "")
73         }
74         names(ar.new) = name
75         para = c(ar.new, intercept = mu.new)
76         print(as.symbol("Number of iterations exceeded options."))
77         break
78     }
79 }
80 }else if (order.ar==0){
81     e = vector("numeric", length = 1+order.ma)
82     for (i in 1:l){
83         e[i+order.ma] = data[i]+
84             initial.ma%%rev(e[i:(i+order.ma-1)])-
85             initial.mu
86     }
87     L.new = sum(e^2)
88     dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
89     dEdMU = matrix(0, ncol = 1+order.ma, nrow = 1)
90     for (i in 1:l){
91         dEdMA[,i+order.ma] =
92             dEdMA[,i:(i+order.ma-1)]%%rev(initial.ma)+
93             rev(e[i:(i+order.ma-1)])
94         dEdMU[i+order.ma] = -1+
95             rev(initial.ma)%dEdMU[i:(i+order.ma-1)]
96     }
97     dEdMA = dEdMA[, (1+order.ma):(1+order.ma)]
98     dEdMU = dEdMU[(1+order.ma):(1+order.ma)]
99     dE = rbind(dEdMA, dEdMU)
100     d = 2*dE%%e[(1+order.ma):(1+order.ma)]
101     G = G+sum(d^2)
102     delta = -d/sqrt(G)*LearningRate
103     p = 0
104     ma.new = initial.ma+delta[1:order.ma]
105     mu.new = initial.mu+delta[1+order.ma]
106     while (T){
107         e = vector("numeric", length = 1+order.ma)
108         for (i in 1:l){
109             e[i+order.ma] = data[i]+
110                 ma.new%%rev(e[i:(i+order.ma-1)])-

```



```

111     mu.new
112   }
113   L = L.new
114   L.new = sum(e^2)
115   if (abs(L-L.new)<tol&sum(d^2)<tol){
116     name = NULL
117     for (i in 1:order.ma){
118       name[i] = paste("ma", as.character(i), sep = "")
119     }
120     names(ma.new) = name
121     para = c(ma.new, intercept = mu.new)
122     print(as.symbol("Local minimum found."))
123     break
124   }
125   dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
126   dEdMU = matrix(0, ncol = 1+order.ma, nrow = 1)
127   for (i in 1:l){
128     dEdMA[,i+order.ma] =
129       dEdMA[,i:(i+order.ma-1)]%*%rev(ma.new)+
130       rev(e[i:(i+order.ma-1)])
131     dEdMU[i+order.ma] = -1+
132       rev(ma.new)%*%dEdMU[i:(i+order.ma-1)]
133   }
134   dE = rbind(dEdMA, dEdMU)
135   dE = dE[(1+order.ma):(1+order.ma)]
136   d = 2*dE%*%e[(1+order.ma):(1+order.ma)]
137   G = G+sum(d^2)
138   delta = -d/sqrt(G)*LearningRate
139   ma.new = ma.new+delta[1:order.ma]
140   mu.new = mu.new+delta[1+order.ma]
141   p = p+1
142   if (p>maxstep){
143     name = NULL
144     for (i in 1:order.ma){
145       name[i] = paste("ma", as.character(i), sep = "")
146     }
147     names(ma.new) = name
148     para = c(ma.new, intercept = mu.new)
149     print(as.symbol("Number of iterations exceeded options."))
150     break
151   }
152 }
153 }else{
154   e = vector("numeric", length = 1+order.ma)
155   for (i in (order.ar+1):l){
156     e[i+order.ma] = data[i]+
157       initial.ma%*%rev(e[i:(i+order.ma-1)])-
158       initial.ar%*%rev(data[(i-order.ar):(i-1)])+

```

```

159     initial.mu*(sum(initial.ar)-1)
160   }
161   L.new = sum(e^2)
162   dEdAR = matrix(0, ncol = 1+order.ma, nrow = order.ar)
163   dEdMU = matrix(0, ncol = 1+order.ma, nrow = 1)
164   dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
165   for (i in (order.ar+1):l){
166     dEdAR[, (i+order.ma)] =
167       dEdAR[, (i:(i+order.ma-1))]%*%rev(initial.ma)+
168       rep( initial.mu, order.ar)-rev(data[(i-order.ar):(i-1)])
169     dEdMU[i+order.ma] = sum(initial.ar)-1+
170       rev( initial.ma)%*%dEdMU[i:(i+order.ma-1)]
171     dEdMA[, (i+order.ma)] =
172       dEdMA[, (i:(i+order.ma-1))]%*%rev(initial.ma)+
173       rev(e[i:(i+order.ma-1)])
174   }
175   dEdAR = dEdAR[, (1+order.ma):(1+order.ma)]
176   dEdMA = dEdMA[, (1+order.ma):(1+order.ma)]
177   dEdMU = dEdMU[(1+order.ma):(1+order.ma)]
178   dE = rbind(dEdAR, dEdMA, dEdMU)
179   d = 2*dE%*%e[(1+order.ma):(1+order.ma)]
180   G = G+sum(d^2)
181   delta = -d/sqrt(G)*LearningRate
182   p = 0
183   ar.new = initial.ar+delta[1:order.ar]
184   ma.new = initial.ma+delta[(1+order.ar):(order.ma+order.ar)]
185   mu.new = initial.mu+delta[1+order.ma+order.ar]
186   while (T){
187     e = vector("numeric", length = 1+order.ma)
188     for (i in (order.ar+1):l){
189       e[i+order.ma] = data[i]+
190         ma.new%*%rev(e[i:(i+order.ma-1)])-
191         ar.new%*%rev(data[(i-order.ar):(i-1)])+
192         mu.new*(sum(ar.new)-1)
193     }
194     L = L.new
195     L.new = sum(e^2)
196     if (abs(L-L.new)<tol&sum(d^2)<tol){
197       name.ar = NULL
198       for (i in 1:order.ar){
199         name.ar[i] = paste("ar", as.character(i), sep = "")
200       }
201       names(ar.new) = name.ar
202       name.ma = NULL
203       for (i in 1:order.ma){
204         name.ma[i] = paste("ma", as.character(i), sep = "")
205       }
206       names(ma.new) = name.ma

```

```

207     para = c(ar.new, ma.new, intercept = mu.new)
208     print(as.symbol("Local minimum found."))
209     break
210   }
211   dEdAR = matrix(0, ncol = 1+order.ma, nrow = order.ar)
212   dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
213   dEdMU = matrix(0, ncol = 1+order.ma, nrow = 1)
214   for (i in (order.ar+1):l){
215     dEdAR[,i+order.ma] =
216       dEdAR[,i:(i+order.ma-1)]%*%rev(ma.new)+
217       rep(mu.new, order.ar)-rev(data[(i-order.ar):(i-1)])
218     dEdMU[i+order.ma] = sum(ar.new)-1+
219       rev(ma.new)%*%dEdMU[i:(i+order.ma-1)]
220     dEdMA[,i+order.ma] =
221       dEdMA[,i:(i+order.ma-1)]%*%rev(ma.new)+
222       rev(e[i:(i+order.ma-1)])
223   }
224   dE = rbind(dEdAR, dEdMA, dEdMU)
225   dE = dE[, (1+order.ma):(1+order.ma)]
226   d = 2*dE%*%e[(1+order.ma):(1+order.ma)]
227   G = G+sum(d^2)
228   delta = -d/sqrt(G)*LearningRate
229   ar.new = ar.new+delta[1:order.ar]
230   ma.new = ma.new+delta[(1+order.ar):(order.ma+order.ar)]
231   mu.new = mu.new+delta[1+order.ma+order.ar]
232   p = p+1
233   if (p>maxstep){
234     name.ar = NULL
235     for (i in 1:order.ar){
236       name.ar[i] = paste("ar", as.character(i), sep = "")
237     }
238     names(ar.new) = name.ar
239     name.ma = NULL
240     for (i in 1:order.ma){
241       name.ma[i] = paste("ma", as.character(i), sep = "")
242     }
243     names(ma.new) = name.ma
244     para = c(ar.new, ma.new, intercept = mu.new)
245     print(as.symbol("Number of iterations exceeded options."))
246     break
247   }
248 }
249 }
250 }else{
251   order.ar = order[1]
252   order.ma = order[2]
253   initial .ar = as.numeric(integer(order[1]))
254   initial .ma = as.numeric(integer(order[2]))

```

```

255 l = length(data)
256 G = 0
257 if (order.ma==0){
258   e = vector("numeric", length = l)
259   for (i in (order.ar+1):l){
260     e[i] = data[i]-
261       initial.ar%%rev(data[(i-order.ar):(i-1)])
262   }
263   L.new = sum(e^2)
264   dEdAR = matrix(0, ncol = l, nrow = order.ar)
265   for (i in (order.ar+1):l){
266     dEdAR[,i] = -rev(data[(i-order.ar):(i-1)])
267   }
268   dE = dEdAR
269   d = 2*dE%%e
270   G = G+sum(d^2)
271   delta = -d/sqrt(G)*LearningRate
272   p = 0
273   ar.new = initial.ar+delta[1:order.ar]
274   while (T){
275     e = vector("numeric", length = l)
276     for (i in (order.ar+1):l){
277       e[i] = data[i]-
278         ar.new%%rev(data[(i-order.ar):(i-1)])
279     }
280     L = L.new
281     L.new = sum(e^2)
282     if (abs(L-L.new)<tol&sum(d^2)<tol){
283       name = NULL
284       for (i in 1:order.ar){
285         name[i] = paste("ar", as.character(i), sep = "")
286       }
287       names(ar.new) = name
288       para = ar.new
289       print(as.symbol("Local minimum found."))
290       break
291     }
292     dEdAR = matrix(0, ncol = l, nrow = order.ar)
293     for (i in (order.ar+1):l){
294       dEdAR[,i] = -rev(data[(i-order.ar):(i-1)])
295     }
296     dE = dEdAR
297     d = 2*dE%%e
298     G = G+sum(d^2)
299     delta = -d/sqrt(G)*LearningRate
300     ar.new = ar.new+delta[1:order.ar]
301     p = p+1
302     if (p>maxstep){

```

```

303     name = NULL
304     for (i in 1:order.ar){
305         name[i] = paste("ar", as.character(i), sep = "")
306     }
307     names(ar.new) = name
308     para = ar.new
309     print(as.symbol("Number of iterations exceeded options."))
310     break
311 }
312 }
313 }else if (order.ar==0){
314     e = vector("numeric", length = l+order.ma)
315     for (i in 1:l){
316         e[i+order.ma] = data[i]+
317             initial.ma%%rev(e[i:(i+order.ma-1)])
318     }
319     L.new = sum(e^2)
320     dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
321     for (i in 1:l){
322         dEdMA[,i+order.ma] =
323             dEdMA[,i:(i+order.ma-1)]%%rev(initial.ma)+
324             rev(e[i:(i+order.ma-1)])
325     }
326     dEdMA = dEdMA[, (1+order.ma):(l+order.ma)]
327     dE = dEdMA
328     d = 2*dE%%e[(1+order.ma):(l+order.ma)]
329     G = G+sum(d^2)
330     delta = -d/sqrt(G)*LearningRate
331     p = 0
332     ma.new = initial.ma+delta[1:order.ma]
333     while (T){
334         e = vector("numeric", length = l+order.ma)
335         for (i in 1:l){
336             e[i+order.ma] = data[i]+
337                 ma.new%%rev(e[i:(i+order.ma-1)])
338         }
339         L = L.new
340         L.new = sum(e^2)
341         if (abs(L-L.new)<tol&sum(d^2)<tol){
342             name = NULL
343             for (i in 1:order.ma){
344                 name[i] = paste("ma", as.character(i), sep = "")
345             }
346             names(ma.new) = name
347             para = ma.new
348             print(as.symbol("Local minimum found."))
349             break
350         }

```

```

351     dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
352     for (i in 1:l){
353         dEdMA[, (i+order.ma)] =
354             dEdMA[, (i:(i+order.ma-1))] %*% rev(ma.new) +
355             rev(e[i:(i+order.ma-1)])
356     }
357     dE = dEdMA
358     dE = dE[, (1+order.ma):(1+order.ma)]
359     d = 2*dE %*% e[(1+order.ma):(1+order.ma)]
360     G = G + sum(d^2)
361     delta = -d/sqrt(G)*LearningRate
362     ma.new = ma.new + delta[1:order.ma]
363     p = p+1
364     if (p>maxstep){
365         name = NULL
366         for (i in 1:order.ma){
367             name[i] = paste("ma", as.character(i), sep = "")
368         }
369         names(ma.new) = name
370         para = ma.new
371         print(as.symbol("Number of iterations exceeded options."))
372         break
373     }
374 }
375 } else {
376     e = vector("numeric", length = 1+order.ma)
377     for (i in (order.ar+1):l){
378         e[i+order.ma] = data[i] +
379             initial.ma %*% rev(e[i:(i+order.ma-1)]) -
380             initial.ar %*% rev(data[(i-order.ar):(i-1)])
381     }
382     L.new = sum(e^2)
383     dEdAR = matrix(0, ncol = 1+order.ma, nrow = order.ar)
384     dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
385     for (i in (order.ar+1):l){
386         dEdAR[, (i+order.ma)] =
387             dEdAR[, (i:(i+order.ma-1))] %*% rev(initial.ma) -
388             rev(data[(i-order.ar):(i-1)])
389         dEdMA[, (i+order.ma)] =
390             dEdMA[, (i:(i+order.ma-1))] %*% rev(initial.ma) +
391             rev(e[i:(i+order.ma-1)])
392     }
393     dEdAR = dEdAR[, (1+order.ma):(1+order.ma)]
394     dEdMA = dEdMA[, (1+order.ma):(1+order.ma)]
395     dE = rbind(dEdAR, dEdMA)
396     d = 2*dE %*% e[(1+order.ma):(1+order.ma)]
397     G = G + sum(d^2)
398     delta = -d/sqrt(G)*LearningRate

```

```

399     p = 0
400     ar.new = initial.ar+delta[1:order.ar]
401     ma.new = initial.ma+delta[(1+order.ar):(order.ma+order.ar)]
402     while (T){
403         e = vector("numeric", length = 1+order.ma)
404         for (i in (order.ar+1):1){
405             e[i+order.ma] = data[i]+
406                 rev(e[i:(i+order.ma-1)])%*%ma.new-
407                 ar.new%*%rev(data[(i-order.ar):(i-1)])
408         }
409         L = L.new
410         L.new = sum(e^2)
411         if (abs(L-L.new)<tol&sum(d^2)<tol){
412             name.ar = NULL
413             for (i in 1:order.ar){
414                 name.ar[i] = paste("ar", as.character(i), sep = "")
415             }
416             names(ar.new) = name.ar
417             name.ma = NULL
418             for (i in 1:order.ma){
419                 name.ma[i] = paste("ma", as.character(i), sep = "")
420             }
421             names(ma.new) = name.ma
422             para = c(ar.new, ma.new)
423             print(as.symbol("Local minimum found."))
424             break
425         }
426         dEdAR = matrix(0, ncol = 1+order.ma, nrow = order.ar)
427         dEdMA = matrix(0, ncol = 1+order.ma, nrow = order.ma)
428         for (i in (order.ar+1):1){
429             dEdAR[,i+order.ma] =
430                 dEdAR[,i:(i+order.ma-1)]%*%rev(ma.new)-
431                 rev(data[(i-order.ar):(i-1)])
432             dEdMA[,i+order.ma] =
433                 dEdMA[,i:(i+order.ma-1)]%*%rev(ma.new)+
434                 rev(e[i:(i+order.ma-1)])
435         }
436         dE = rbind(dEdAR, dEdMA)
437         dE = dE[(1+order.ma):(1+order.ma)]
438         d = 2*dE%*%e[(1+order.ma):(1+order.ma)]
439         G = G+sum(d^2)
440         delta = -d/sqrt(G)*LearningRate
441         ar.new = ar.new+delta[1:order.ar]
442         ma.new = ma.new+delta[(1+order.ar):(order.ma+order.ar)]
443         p = p+1
444         if (p>maxstep){
445             name.ar = NULL
446             for (i in 1:order.ar){

```

```

447     name.ar[i] = paste("ar", as.character(i), sep = "")
448   }
449   names(ar.new) = name.ar
450   name.ma = NULL
451   for (i in 1:order.ma){
452     name.ma[i] = paste("ma", as.character(i), sep = "")
453   }
454   names(ma.new) = name.ma
455   para = c(ar.new, ma.new)
456   print(as.symbol("Number of iterations exceeded options."))
457   break
458   }
459 }
460 }
461 }
462 return(para)
463 }

```

### 3.3 极大似然估计

以下为对AR(1)模型参数进行极大似然估计的函数

```

1 mle.ar1 = function(data, intercept = T,
2                     tol = 1e-05,
3                     maxstep = 1e+04,
4                     LearningRate = 1){
5   #arima,order=c(1,0,0),method="ML"
6   if (intercept==T){
7     initial.ar = 0
8     initial.mu = 0
9     initial.sigma2 = 1
10    l = length(data)
11    G = 0
12    e = vector("numeric", length = l)
13    for (i in 2:l){
14      e[i] = data[i]-
15        initial.ar*data[i-1]+initial.mu*(initial.ar-1)
16    }
17    L.new = -(sum(e^2)+
18              (1-initial.ar^2)*(data[1]-initial.mu)^2)/2/initial.sigma2+
19            -l/2*log(2*pi)-l/2*log(initial.sigma2)+l/2*log(1-initial.ar^2)
20    L.new = -L.new
21    dEdAR = matrix(0, ncol = l, nrow = 1)
22    dEdMU = matrix(0, ncol = l, nrow = 1)
23    for (i in 2:l){
24      dEdAR[i] = initial.mu-data[i-1]
25      dEdMU[i] = initial.ar-1

```



```

26 }
27 dLdAR = -initial.ar/(1-initial.ar^2)+
28   initial .ar*(data[1]- initial .mu)^2/initial.sigma2-
29   dEdAR%%e/initial.sigma2
30 dLdMU = (1-initial.ar^2)*(data[1]-initial.mu)/initial.sigma2-
31   dEdMU%%e/initial.sigma2
32 dLdS2 = -1/2/initial.sigma2+
33   (sum(e^2)+(1-initial.ar^2)*(data[1]- initial .mu)^2)/2/initial .sigma2^2
34 dL = c(-dLdAR, -dLdMU, -dLdS2)
35 G = G+sum(dL^2)
36 delta = -dL/sqrt(G)*LearningRate
37 ar.new = initial.ar+delta[1]
38 mu.new = initial.mu+delta[2]
39 s2.new = initial .sigma2+delta[3]
40 p = 0
41 while (T){
42   e = vector("numeric", length = 1)
43   for (i in 2:1){
44     e[i] = data[i]-ar.new*data[i-1]+mu.new*(ar.new-1)
45   }
46   L = L.new
47   L.new = -(sum(e^2)+
48     (1-ar.new^2)*(data[1]-mu.new)^2)/2/s2.new+
49     -1/2*log(2*pi)-1/2*log(s2.new)+1/2*log(1-ar.new^2)
50   L.new = -L.new
51   if (abs(L-L.new)<tol&&sum(dL^2)<tol){
52     para = c(ar1 = ar.new, intercept = mu.new)
53     print(as.symbol("Local minimum found. "))
54     break
55   }
56   dEdAR = matrix(0, ncol = 1, nrow = 1)
57   dEdMU = matrix(0, ncol = 1, nrow = 1)
58   for (i in 2:1){
59     dEdAR[,i] = mu.new-data[i-1]
60     dEdMU[i] = ar.new-1
61   }
62   dLdAR = -ar.new/(1-ar.new^2)+
63     ar.new*(data[1]-mu.new)^2/s2.new-
64     dEdAR%%e/s2.new
65   dLdMU = (1-ar.new^2)*(data[1]-mu.new)/s2.new-
66     dEdMU%%e/s2.new
67   dLdS2 = -1/2/s2.new+
68     (sum(e^2)+(1-initial.ar^2)*(data[1]- initial .mu)^2)/2/s2.new^2
69   dL = c(-dLdAR, -dLdMU, -dLdS2)
70   G = G+sum(dL^2)
71   delta = -dL/sqrt(G)*LearningRate
72   ar.new = ar.new+delta[1]
73   mu.new = mu.new+delta[2]

```

```

74     s2.new = s2.new+delta[3]
75     p = p+1
76     if (p>maxstep){
77         para = c(ar1 = ar.new, intercept = mu.new)
78         print(as.symbol("Number of iterations exceeded options."))
79         break
80     }
81 }
82 }else{
83     initial.ar = 0
84     initial.sigma2 = 1
85     l = length(data)
86     G = 0
87     e = vector("numeric", length = l)
88     for (i in 2:l){
89         e[i] = data[i] - initial.ar*data[i-1]
90     }
91     L.new = -(sum(e^2)+
92             (1-initial.ar^2)*data[1]^2)/2/initial.sigma2+
93             -1/2*log(2*pi)-1/2*log(initial.sigma2)+1/2*log(1-initial.ar^2)
94     L.new = -L.new
95     dEdAR = matrix(0, ncol = 1, nrow = 1)
96     for (i in 2:l){
97         dEdAR[i] = -data[i-1]
98     }
99     dLdAR = -initial.ar/(1-initial.ar^2)+
100     initial.ar*data[1]^2/initial.sigma2-
101     dEdAR%%e/initial.sigma2
102     dLdS2 = -1/2/initial.sigma2+
103     (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/initial.sigma2^2
104     dL = c(-dLdAR, -dLdS2)
105     G = G+sum(dL^2)
106     delta = -dL/sqrt(G)*LearningRate
107     ar.new = initial.ar+delta[1]
108     s2.new = initial.sigma2+delta[2]
109     p = 0
110     while (T){
111         e = vector("numeric", length = l)
112         for (i in 2:l){
113             e[i] = data[i] - ar.new*data[i-1]
114         }
115         L = L.new
116         L.new = -(sum(e^2)+
117                 (1-ar.new^2)*data[1]^2)/2/s2.new+
118                 -1/2*log(2*pi)-1/2*log(s2.new)+1/2*log(1-ar.new^2)
119         L.new = -L.new
120         if (abs(L-L.new)<tol&&sum(dL^2)<tol){
121             para = c(ar1 = ar.new)

```

```

122     print(as.symbol("Local minimum found.))
123     break
124 }
125 dEdAR = matrix(0, ncol = 1, nrow = 1)
126 for (i in 2:l){
127     dEdAR[,i] = -data[i-1]
128 }
129 dLdAR = -ar.new/(1-ar.new^2)+
130     ar.new*data[1]^2/s2.new-
131     dEdAR%%e/s2.new
132 dLdS2 = -1/2/s2.new+
133     (sum(e^2)+(1-initial.ar^2)*data[1]^2)/2/s2.new^2
134 dL = c(-dLdAR, -dLdS2)
135 G = G+sum(dL^2)
136 delta = -dL/sqrt(G)*LearningRate
137 ar.new = ar.new+delta[1]
138 s2.new = s2.new+delta[2]
139 p = p+1
140 if (p>maxstep){
141     para = c(ar1 = ar.new)
142     print(as.symbol("Number of iterations exceeded options.))
143     break
144 }
145 }
146 }
147 return(para)
148 }

```

### 3.4 无条件最小二乘

以下为对AR(1)模型参数进行无条件最小二乘估计的函数

```

1 ulse.ar1 = function(data, intercept = T,
2     tol = 1e-05,
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 2:1){
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 = 1)
33   for (i in 2:1){
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 2:1){
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:l){
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:l){
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 = l)
85         for (i in 2:l){
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:l){
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 }
```

3.5 使用说明

(1) 以上所有函数的输入参数的名称与含义如下表：

| 输入参数名称       | 含义                                       |
|--------------|--|
| data         | 给定的时间序列数据                                |
| order        | me.ar中是ar模型的阶数；cls.arma中是arma模型的阶数，为二元向量 |
| intercept    | 确定模型中是否包含均值参数（默认为TRUE）                   |
| tol          | 跳出迭代的阈值（默认为1e-05）                        |
| maxstep      | 最大迭代次数（默认为1e+04）                         |
| LearningRate | 学习率（默认为1）                                |

表 1: 函数的输入参数的名称与含义

- (2) 函数的输出参数为模型参数的估计值。
- (3) 矩估计函数需要导入自定义函数acfun。
- (4) intercept是布尔型变量。如果为TRUE，则模型中包含均值参数。

4 与库函数的对比

首先导入TSA程辑包中的部分时间序列数据集，其信息如下：

| 数据集名称    | 生成数据模型的参数值                      |
|----------|---------------------------------|
| ar1.s    | $\phi = 0.9$                    |
| ar2.s    | $\phi_1 = 1.5, \phi_2 = -0.75$  |
| ma1.1.s  | $\theta = 0.9$                  |
| ma2.s    | $\theta_1 = 1, \theta_2 = -0.6$ |
| arma11.s | $\phi = 0.6, \theta = -0.3$     |

表 2: 时间序列数据集的名称与生成数据模型的参数

接下来对这些数据集来自的模型进行参数估计。

## 4.1 数值解一致的算法

以下函数的估计结果分别相同（只是可能在精度上有些许区别）

(1) `cls.arma(data,order)`与`arima(data,order,method="CSS")`

以下对`ar2.s`数据集进行测试

```

1 > clse.arma(ar2.s,c(2,0),LearningRate = 1)
2 Local minimum found.
3      ar1      ar2  intercept
4  1.5137151 -0.8049925 0.2631830
5 > arima(ar2.s,c(2,0,0),method="CSS")
6
7 Call:
8 arima(x = ar2.s, order = c(2, 0, 0), method = "CSS")
9
10 Coefficients :
11      ar1      ar2  intercept
12  1.5137  -0.8050   0.2637
13 s.e.  0.0550  0.0549   0.2927
14
15 sigma^2 estimated as 0.8713: part log likelihood = -162.01

```

```

1 > clse.arma(ar2.s,c(2,0),LearningRate = 1,intercept = F)
2 Local minimum found.
3      ar1      ar2
4  1.5152597 -0.8046584
5 > arima(ar2.s,c(2,0,0),method="CSS",include.mean = F)
6
7 Call:
8 arima(x = ar2.s, order = c(2, 0, 0), include.mean = F, method = "CSS")
9
10 Coefficients :
11      ar1      ar2
12  1.5153  -0.8047
13 s.e.  0.0552  0.0551
14
15 sigma^2 estimated as 0.8772: part log likelihood = -162.41

```

以下对`ma2.s`数据集进行测试

```

1 > clse.arma(ma2.s,c(0,2),LearningRate = 0.1)
2 Local minimum found.
3      ma1      ma2  intercept
4  1.0559739 -0.5722273 0.1352184
5 > arima(ma2.s,c(0,0,2),method="CSS")
6
7 Call:
8 arima(x = ma2.s, order = c(0, 0, 2), method = "CSS")

```

```

9
10 Coefficients :
11          ma1      ma2 intercept
12      -1.0560 0.5723      0.1352
13 s.e.   0.0873 0.0863      0.0511
14
15 sigma^2 estimated as 1.184: part log likelihood = -180.42

```

```

1 > clse.arma(ma2.s,c(0,2),LearningRate = 0.1,intercept = F)
2 Local minimum found.
3          ma1      ma2
4      1.0300179 -0.5775751
5 > arima(ma2.s,c(0,0,2),method="CSS",include.mean = F)
6
7 Call:
8 arima(x = ma2.s, order = c(0, 0, 2), include.mean = F, method = "CSS")
9
10 Coefficients :
11          ma1      ma2
12      -1.0301 0.5776
13 s.e.   0.0930 0.0844
14
15 sigma^2 estimated as 1.25: part log likelihood = -183.64

```

以下对arma11.s数据集进行测试

```

1 > clse.arma(arma11.s,c(1,1),LearningRate = 0.1)
2 Local minimum found.
3          ar1      ma1 intercept
4      0.5585807 -0.3668805 0.3923015
5 > arima(arma11.s,c(1,0,1),method="CSS")
6
7 Call:
8 arima(x = arma11.s, order = c(1, 0, 1), method = "CSS")
9
10 Coefficients :
11          ar1      ma1 intercept
12      0.5586 0.3669      0.3928
13 s.e.   0.1219 0.1564      0.3380
14
15 sigma^2 estimated as 1.199: part log likelihood = -150.98

```

```

1 > clse.arma(arma11.s,c(1,1),LearningRate = 0.1,intercept = F)
2 Local minimum found.
3          ar1      ma1
4      0.5874466 -0.3471722
5 > arima(arma11.s,c(1,0,1),method="CSS",include.mean = F)
6

```



```

7 Call:
8 arima(x = arma11.s, order = c(1, 0, 1), include.mean = F, method = "CSS")
9
10 Coefficients :
11      ar1      ma1
12    0.5875 0.3471
13 s.e. 0.1177 0.1567
14
15 sigma^2 estimated as 1.215: part log likelihood = -151.62

```

(2) mle.ar1(data)与arima(data,order=c(1,0,0),method="ML")

以下对ar1.s数据集进行测试

```

1 > mle.ar1(ar1.s)
2 Local minimum found.
3      ar1 intercept
4 0.8922657 1.2520913
5 > arima(ar1.s,c(1,0,0),method="ML")
6
7 Call:
8 arima(x = ar1.s, order = c(1, 0, 0), method = "ML")
9
10 Coefficients :
11      ar1 intercept
12    0.8924    1.2631
13 s.e. 0.0598    1.1399

```

```

1 > mle.ar1(ar1.s,intercept = F)
2 Local minimum found.
3      ar1
4 0.9244036
5 > arima(ar1.s,c(1,0,0),method="ML",include.mean = F)
6
7 Call:
8 arima(x = ar1.s, order = c(1, 0, 0), include.mean = F, method = "ML")
9
10 Coefficients :
11      ar1
12    0.9250
13 s.e. 0.0423
14
15 sigma^2 estimated as 1.048: log likelihood = -87.52, aic = 177.04

```

(3) me.ar(data,order)与ar.yw(data,order.max)

以下对ar2.s数据集进行测试

```

1 > me.ar(ar2.s,2)

```

```

2      ar1      ar2
3  1.4694476 -0.7646034
4 > ar.yw(ar2.s,order.max = 2)
5
6 Call:
7 ar.yw.default(x = ar2.s, order.max = 2)
8
9 Coefficients :
10      1      2
11  1.4694 -0.7646
12
13 Order selected 2  sigma^2 estimated as 1.051

```

## 4.2 数值解不一致的算法

以下函数的估计结果分别有一定差异（可能是函数实现的算法不一致而导致的结果）

(1) `uls.ar1(data)`与`arima(data,order=c(1,0,0))`。考虑到`arima`函数提供的`method`选项只有“CSS”,“ML”,“CSS-ML”三种，故认为如果可以，则“CSS-ML”应该对应了无条件最小二乘法。以下对`ar1.s`数据集进行测试

```

1 > ulse.ar1(ar1.s)
2 Local minimum found.
3      ar1 intercept
4  0.8610367 1.4113062
5 > arima(ar1.s,c(1,0,0),method = "CSS-ML")
6
7 Call:
8 arima(x = ar1.s, order = c(1, 0, 0), method = "CSS-ML")
9
10 Coefficients :
11      ar1 intercept
12    0.8924    1.2630
13 s.e.  0.0598    1.1399
14
15 sigma^2 estimated as 1.041: log likelihood = -87.13, aic = 178.26

```

```

1 > ulse.ar1(ar1.s,intercept = F)
2 Local minimum found.
3      ar1
4  0.9283047
5 > arima(ar1.s,c(1,0,0),method = "CSS-ML",include.mean = F)
6
7 Call:

```

```
8 arima(x = ar1.s, order = c(1, 0, 0), include.mean = F, method = "CSS-ML")
9
10 Coefficients :
11      ar1
12    0.9250
13 s.e.  0.0423
14
15 sigma^2 estimated as 1.048: log likelihood = -87.52, aic = 177.04
```

结果表明两者的算法应该不一致。

(2) `me.arma11(data)`与`arima(data,order=c(1,0,0))`。在`arima`函数的`method`选项中没有矩估计相关的方法种类，而R语言中自带的时间序列参数估计函数只有`arima`一个可以估计`arma`序列参数，故理论上`me.arma11`就应该找不到对应的库函数。

```
1 > me.arma11(arma11.s)
2      ar1      ma1
3 0.6377807 -0.2038076
4 > arima(arma11.s,c(1,0,1),include.mean = F)
5
6 Call:
7 arima(x = arma11.s, order = c(1, 0, 1), include.mean = F)
8
9 Coefficients :
10      ar1      ma1
11    0.5877  0.3417
12 s.e.  0.1161  0.1579
13
14 sigma^2 estimated as 1.207: log likelihood = -151.76, aic = 307.52
```

结果表明两者的算法应该不一致。