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

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

解这个方程就可以解得$\{\phi_1, \phi_2, \cdots, \phi_p\}$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

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

其中$f(\boldsymbol{x})$是$\boldsymbol{R}^n$上具有一阶连续偏导的函数。那么如果选取适当的初值$\boldsymbol{x}^{(0)}$，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$f(\boldsymbol{x}) \in \boldsymbol{R}^n$; 梯度函数$g(\boldsymbol{x}) = \nabla f(x)$; 初始迭代值$\boldsymbol{x}^{(0)} \in \boldsymbol{R}^n$; 计算精度$\epsilon$; 学习率$\lambda$

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

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

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

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

自适应梯度下降法中则是这样一种情况

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

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：

$$\begin{aligned} Y_t =& \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} + \\ & e_t - \theta_1 e_{t-1} - \theta_2 e_{t-2} - \cdots - \theta_q e_{t-q} \end{aligned}$$

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：

$$\begin{aligned} Y_t-\mu =& \phi_1(Y_{t-1}-\mu)+\cdots+\phi_p(Y_{t-p}-\mu)+ \\ & e_t-\theta_1 e_{t-1}-\cdots-\theta_q e_{t-q} \\ i.e. \quad Y_t =& \phi_1 Y_{t-1}+\cdots+\phi_p Y_{t-p}+ \\ & e_t-\theta_1 e_{t-1}-\cdots-\theta_q e_{t-q}+(1-\sum_p \phi_i)\mu \end{aligned}$$

e_t的表达式变为

$$\begin{aligned} e_t(\boldsymbol{\phi},\boldsymbol{\theta},\mu) =& Y_t-\phi_1 Y_{t-1}-\cdots-\phi_p Y_{t-p}+ \\ & \theta_1 e_{t-1}+\cdots+\theta_q e_{t-q}-(1-\sum_p \phi_i)\mu \end{aligned}$$

最小化的目标函数变为$S_c(\boldsymbol{\phi},\boldsymbol{\theta},\mu)=\sum_{t=p+1}^n e_t^2$，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$S_c(\boldsymbol{\phi},\boldsymbol{\theta},\mu)$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：

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

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：

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

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

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

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

```
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

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

```
      RHO = rbind(c(1, acfval[1:(i-1)]), cbind(acfval[1:(i-1)], RHO))
    }
  }
  phi = as.numeric(qr.solve(RHO, acfval))
  name = NULL
  for (i in 1:order){
    name[i] = paste("ar", as.character(i), sep = "")
  }
  names(phi) = name
  para = phi
  return(para)
}
```

以下为对ARMA(1,1)模型参数进行矩估计的函数

```
me.arma11 = function(data){
  result.acf = acfun(data = data-mean(data), lag.max = 2, plot = FALSE)
  acfval = result.acf$acfval
  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)
}
```

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

```
clse.arma = function(data, order, intercept = T,
                     tol = 1e-04,
                     maxstep = 1e+04,
                     LearningRate = 0.1){
  #arima,method="CSS"
  if (intercept==T){
    order.ar = order[1]
    order.ma = order[2]
    initial.ar = as.numeric(integer(order[1]))
    initial.ma = as.numeric(integer(order[2]))
    initial.mu = 0
    l = length(data)
    G = 0
    if (order.ma==0){
```

```r
    e = vector("numeric", length = l)
    for (i in (order.ar+1):l){
      e[i] = data[i]-
        initial.ar%*%rev(data[(i-order.ar):(i-1)])+
        initial.mu*(sum(initial.ar)-1)
    }
    L.new = sum(e^2)
    dEdAR = matrix(0, ncol = l, nrow = order.ar)
    dEdMU = matrix(0, ncol = l, nrow = 1)
    for (i in (order.ar+1):l){
      dEdAR[,i] = rep(initial.mu, order.ar)-
        rev(data[(i-order.ar):(i-1)])
      dEdMU[i] = sum(initial.ar)-1
    }
    dE = rbind(dEdAR, dEdMU)
    d = 2*dE%*%e
    G = G+sum(d^2)
    delta = -d/sqrt(G)*LearningRate
    p = 0
    ar.new = initial.ar+delta[1:order.ar]
    mu.new = initial.mu+delta[1+order.ar]
    while (T){
      e = vector("numeric", length = l)
      for (i in (order.ar+1):l){
        e[i] = data[i]-
          ar.new%*%rev(data[(i-order.ar):(i-1)])+
          mu.new*(sum(ar.new)-1)
      }
      L = L.new
      L.new = sum(e^2)
      if (abs(L-L.new)<tol&sum(d^2)<tol){
        name = NULL
        for (i in 1:order.ar){
          name[i] = paste("ar", as.character(i), sep = "")
        }
        names(ar.new) = name
        para = c(ar.new, intercept = mu.new)
        print(as.symbol("Local minimum found."))
        break
      }
      dEdAR = matrix(0, ncol = l, nrow = order.ar)
      dEdMU = matrix(0, ncol = l, nrow = 1)
      for (i in (order.ar+1):l){
        dEdAR[,i] =
          rep(mu.new, order.ar)-rev(data[(i-order.ar):(i-1)])
        dEdMU[i] = sum(ar.new)-1
      }
      dE = rbind(dEdAR, dEdMU)
```

```r
      d = 2*dE%*%e
      G = G+sum(d^2)
      delta = -d/sqrt(G)*LearningRate
      ar.new = ar.new+delta[1:order.ar]
      mu.new = mu.new+delta[1+order.ar]
      p = p+1
      if (p>maxstep){
        name = NULL
        for (i in 1:order.ar){
          name[i] = paste("ar", as.character(i), sep = "")
        }
        names(ar.new) = name
        para = c(ar.new, intercept = mu.new)
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }else if (order.ar==0){
    e = vector("numeric", length = l+order.ma)
    for (i in 1:l){
      e[i+order.ma] = data[i]+
         initial.ma%*%rev(e[i:(i+order.ma-1)])-
         initial.mu
    }
    L.new = sum(e^2)
    dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
    dEdMU = matrix(0, ncol = l+order.ma, nrow = 1)
    for (i in 1:l){
      dEdMA[,(i+order.ma)] =
        dEdMA[,(i:(i+order.ma-1))]%*%rev(initial.ma)+
        rev(e[i:(i+order.ma-1)])
      dEdMU[i+order.ma] = -1+
        rev(initial.ma)%*%dEdMU[i:(i+order.ma-1)]
    }
    dEdMA = dEdMA[,(1+order.ma):(l+order.ma)]
    dEdMU = dEdMU[(1+order.ma):(l+order.ma)]
    dE = rbind(dEdMA, dEdMU)
    d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
    G = G+sum(d^2)
    delta = -d/sqrt(G)*LearningRate
    p = 0
    ma.new = initial.ma+delta[1:order.ma]
    mu.new = initial.mu+delta[1+order.ma]
    while (T){
      e = vector("numeric", length = l+order.ma)
      for (i in 1:l){
        e[i+order.ma] = data[i]+
          ma.new%*%rev(e[i:(i+order.ma-1)])-
```

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

```
            initial.mu*(sum(initial.ar)-1)
    }
    L.new = sum(e^2)
    dEdAR = matrix(0, ncol = l+order.ma, nrow = order.ar)
    dEdMU = matrix(0, ncol = l+order.ma, nrow = 1)
    dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
    for (i in (order.ar+1):l){
      dEdAR[,(i+order.ma)] =
        dEdAR[,(i:(i+order.ma-1))]%*%rev(initial.ma)+
        rep(initial.mu, order.ar)-rev(data[(i-order.ar):(i-1)])
      dEdMU[i+order.ma] = sum(initial.ar)-1+
        rev(initial.ma)%*%dEdMU[i:(i+order.ma-1)]
      dEdMA[,(i+order.ma)] =
        dEdMA[,(i:(i+order.ma-1))]%*%rev(initial.ma)+
        rev(e[i:(i+order.ma-1)])
    }
    dEdAR = dEdAR[,(1+order.ma):(l+order.ma)]
    dEdMA = dEdMA[,(1+order.ma):(l+order.ma)]
    dEdMU = dEdMU[(1+order.ma):(l+order.ma)]
    dE = rbind(dEdAR, dEdMA, dEdMU)
    d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
    G = G+sum(d^2)
    delta = -d/sqrt(G)*LearningRate
    p = 0
    ar.new = initial.ar+delta[1:order.ar]
    ma.new = initial.ma+delta[(1+order.ar):(order.ma+order.ar)]
    mu.new = initial.mu+delta[1+order.ma+order.ar]
    while (T){
      e = vector("numeric", length = l+order.ma)
      for (i in (order.ar+1):l){
        e[i+order.ma] = data[i]+
          ma.new%*%rev(e[i:(i+order.ma-1)])-
          ar.new%*%rev(data[(i-order.ar):(i-1)])+
          mu.new*(sum(ar.new)-1)
      }
      L = L.new
      L.new = sum(e^2)
      if (abs(L-L.new)<tol&sum(d^2)<tol){
        name.ar = NULL
        for (i in 1:order.ar){
          name.ar[i] = paste("ar", as.character(i), sep = "")
        }
        names(ar.new) = name.ar
        name.ma = NULL
        for (i in 1:order.ma){
          name.ma[i] = paste("ma", as.character(i), sep = "")
        }
        names(ma.new) = name.ma
```

```
          para = c(ar.new, ma.new, intercept = mu.new)
          print(as.symbol("Local minimum found."))
          break
        }
        dEdAR = matrix(0, ncol = l+order.ma, nrow = order.ar)
        dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
        dEdMU = matrix(0, ncol = l+order.ma, nrow = 1)
        for (i in (order.ar+1):l){
          dEdAR[,(i+order.ma)] =
            dEdAR[,(i:(i+order.ma-1))]%*%rev(ma.new)+
            rep(mu.new, order.ar)-rev(data[(i-order.ar):(i-1)])
          dEdMU[i+order.ma] = sum(ar.new)-1+
            rev(ma.new)%*%dEdMU[i:(i+order.ma-1)]
          dEdMA[,(i+order.ma)] =
            dEdMA[,(i:(i+order.ma-1))]%*%rev(ma.new)+
            rev(e[i:(i+order.ma-1)])
        }
        dE = rbind(dEdAR, dEdMA, dEdMU)
        dE = dE[,(1+order.ma):(l+order.ma)]
        d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
        G = G+sum(d^2)
        delta = -d/sqrt(G)*LearningRate
        ar.new = ar.new+delta[1:order.ar]
        ma.new = ma.new+delta[(1+order.ar):(order.ma+order.ar)]
        mu.new = mu.new+delta[1+order.ma+order.ar]
        p = p+1
        if (p>maxstep){
          name.ar = NULL
          for (i in 1:order.ar){
            name.ar[i] = paste("ar", as.character(i), sep = "")
          }
          names(ar.new) = name.ar
          name.ma = NULL
          for (i in 1:order.ma){
            name.ma[i] = paste("ma", as.character(i), sep = "")
          }
          names(ma.new) = name.ma
          para = c(ar.new, ma.new, intercept = mu.new)
          print(as.symbol("Number of iterations exceeded options."))
          break
        }
      }
    }
  }else{
    order.ar = order[1]
    order.ma = order[2]
    initial.ar = as.numeric(integer(order[1]))
    initial.ma = as.numeric(integer(order[2]))
```

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

```
        name = NULL
        for (i in 1:order.ar){
          name[i] = paste("ar", as.character(i), sep = "")
        }
        names(ar.new) = name
        para = ar.new
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }else if (order.ar==0){
    e = vector("numeric", length = l+order.ma)
    for (i in 1:l){
      e[i+order.ma] = data[i]+
         initial.ma%*%rev(e[i:(i+order.ma-1)])
    }
    L.new = sum(e^2)
    dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
    for (i in 1:l){
      dEdMA[,(i+order.ma)] =
        dEdMA[,(i:(i+order.ma-1))]%*%rev(initial.ma)+
        rev(e[i:(i+order.ma-1)])
    }
    dEdMA = dEdMA[,(1+order.ma):(l+order.ma)]
    dE = dEdMA
    d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
    G = G+sum(d^2)
    delta = -d/sqrt(G)*LearningRate
    p = 0
    ma.new = initial.ma+delta[1:order.ma]
    while (T){
      e = vector("numeric", length = l+order.ma)
      for (i in 1:l){
        e[i+order.ma] = data[i]+
          ma.new%*%rev(e[i:(i+order.ma-1)])
      }
      L = L.new
      L.new = sum(e^2)
      if (abs(L-L.new)<tol&sum(d^2)<tol){
        name = NULL
        for (i in 1:order.ma){
          name[i] = paste("ma", as.character(i), sep = "")
        }
        names(ma.new) = name
        para = ma.new
        print(as.symbol("Local minimum found."))
        break
      }
```

```
      dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
      for (i in 1:l){
        dEdMA[,(i+order.ma)] =
          dEdMA[,(i:(i+order.ma-1))]%*%rev(ma.new)+
          rev(e[i:(i+order.ma-1)])
      }
      dE = dEdMA
      dE = dE[,(1+order.ma):(l+order.ma)]
      d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
      G = G+sum(d^2)
      delta = -d/sqrt(G)*LearningRate
      ma.new = ma.new+delta[1:order.ma]
      p = p+1
      if (p>maxstep){
        name = NULL
        for (i in 1:order.ma){
          name[i] = paste("ma", as.character(i), sep = "")
        }
        names(ma.new) = name
        para = ma.new
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }else{
    e = vector("numeric", length = l+order.ma)
    for (i in (order.ar+1):l){
      e[i+order.ma] = data[i]+
         initial.ma%*%rev(e[i:(i+order.ma-1)])-
         initial.ar%*%rev(data[(i-order.ar):(i-1)])
    }
    L.new = sum(e^2)
    dEdAR = matrix(0, ncol = l+order.ma, nrow = order.ar)
    dEdMA = matrix(0, ncol = l+order.ma, nrow = order.ma)
    for (i in (order.ar+1):l){
      dEdAR[,(i+order.ma)] =
        dEdAR[,(i:(i+order.ma-1))]%*%rev(initial.ma)-
        rev(data[(i-order.ar):(i-1)])
      dEdMA[,(i+order.ma)] =
        dEdMA[,(i:(i+order.ma-1))]%*%rev(initial.ma)+
        rev(e[i:(i+order.ma-1)])
    }
    dEdAR = dEdAR[,(1+order.ma):(l+order.ma)]
    dEdMA = dEdMA[,(1+order.ma):(l+order.ma)]
    dE = rbind(dEdAR, dEdMA)
    d = 2*dE%*%e[(1+order.ma):(l+order.ma)]
    G = G+sum(d^2)
    delta = -d/sqrt(G)*LearningRate
```

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

```
          name.ar[i] = paste("ar", as.character(i), sep = "")
        }
        names(ar.new) = name.ar
        name.ma = NULL
        for (i in 1:order.ma){
          name.ma[i] = paste("ma", as.character(i), sep = "")
        }
        names(ma.new) = name.ma
        para = c(ar.new, ma.new)
        print(as.symbol("Number of iterations exceeded options."))
        break
      }
    }
  }
 }
 return(para)
}
```

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

```
mle.ar1 = function(data, intercept = T,
                   tol = 1e-05,
                   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 = -l/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 = -l/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]
```

```r
      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)
}
```

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

```
ulse.ar1 = function(data, intercept = T,
                    tol = 1e-05,
                    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)
}
```

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

```
> clse.arma(ar2.s,c(2,0),LearningRate = 1)
Local minimum found.
        ar1         ar2  intercept
 1.5137151 -0.8049925 0.2631830
> arima(ar2.s,c(2,0,0),method="CSS")

Call:
arima(x = ar2.s, order = c(2, 0, 0), method = "CSS")

 Coefficients:
         ar1      ar2   intercept
      1.5137  -0.8050      0.2637
s.e.  0.0550   0.0549      0.2927

sigma^2 estimated as 0.8713:  part log likelihood = -162.01
```

```
> clse.arma(ar2.s,c(2,0),LearningRate = 1,intercept = F)
Local minimum found.
        ar1         ar2
 1.5152597 -0.8046584
> arima(ar2.s,c(2,0,0),method="CSS",include.mean = F)

Call:
arima(x = ar2.s, order = c(2, 0, 0), include.mean = F, method = "CSS")

 Coefficients:
         ar1      ar2
      1.5153  -0.8047
s.e.  0.0552   0.0551

sigma^2 estimated as 0.8772:  part log likelihood = -162.41
```

以下对ma2.s数据集进行测试

```
> clse.arma(ma2.s,c(0,2),LearningRate = 0.1)
Local minimum found.
        ma1         ma2  intercept
 1.0559739 -0.5722273 0.1352184
> arima(ma2.s,c(0,0,2),method="CSS")

Call:
arima(x = ma2.s, order = c(0, 0, 2), method = "CSS")
```

```

Coefficients :
          ma1     ma2  intercept
      -1.0560  0.5723     0.1352
s.e.   0.0873  0.0863     0.0511

sigma^2 estimated as 1.184:  part log likelihood = -180.42
```

```
> clse.arma(ma2.s,c(0,2),LearningRate = 0.1,intercept = F)
Local minimum found.
      ma1        ma2
 1.0300179 -0.5775751
> arima(ma2.s,c(0,0,2),method="CSS",include.mean = F)

Call:
arima(x = ma2.s, order = c(0, 0, 2), include.mean = F, method = "CSS")

Coefficients :
          ma1     ma2
      -1.0301  0.5776
s.e.   0.0930  0.0844

sigma^2 estimated as 1.25:  part log likelihood = -183.64
```

以下对arma11.s数据集进行测试

```
> clse.arma(arma11.s,c(1,1),LearningRate = 0.1)
Local minimum found.
      ar1        ma1  intercept
 0.5585807 -0.3668805 0.3923015
> arima(arma11.s,c(1,0,1),method="CSS")

Call:
arima(x = arma11.s, order = c(1, 0, 1), method = "CSS")

Coefficients :
         ar1     ma1  intercept
      0.5586  0.3669     0.3928
s.e.  0.1219  0.1564     0.3380

sigma^2 estimated as 1.199:  part log likelihood = -150.98
```

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

```

```
Call:
arima(x = arma11.s, order = c(1, 0, 1), include.mean = F, method = "CSS")

 Coefficients:
          ar1     ma1
       0.5875  0.3471
s.e.   0.1177  0.1567

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

```
> mle.ar1(ar1.s)
Local minimum found.
       ar1  intercept
0.8922657 1.2520913
> arima(ar1.s,c(1,0,0),method="ML")

Call:
arima(x = ar1.s, order = c(1, 0, 0), method = "ML")

 Coefficients:
          ar1  intercept
       0.8924     1.2631
s.e.   0.0598     1.1399
```

```
> mle.ar1(ar1.s, intercept = F)
Local minimum found.
       ar1
0.9244036
> arima(ar1.s,c(1,0,0),method="ML",include.mean = F)

Call:
arima(x = ar1.s, order = c(1, 0, 0), include.mean = F, method = "ML")

 Coefficients:
          ar1
       0.9250
s.e.   0.0423

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

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

```
      ar1        ar2
 1.4694476 −0.7646034
> ar.yw(ar2.s,order.max = 2)

Call:
ar.yw.default(x = ar2.s, order.max = 2)

 Coefficients :
      1        2
 1.4694  −0.7646

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

```
> ulse.ar1(ar1.s)
Local minimum found.
      ar1  intercept
0.8610367 1.4113062
> arima(ar1.s,c(1,0,0),method = "CSS-ML")

Call:
arima(x = ar1.s, order = c(1, 0, 0), method = "CSS-ML")

 Coefficients :
          ar1   intercept
      0.8924      1.2630
s.e.  0.0598      1.1399

sigma^2 estimated as 1.041:  log likelihood = −87.13, aic = 178.26
```

```
> ulse.ar1(ar1.s,intercept = F)
Local minimum found.
      ar1
0.9283047
> arima(ar1.s,c(1,0,0),method = "CSS-ML",include.mean = F)

Call:
```

```
arima(x = ar1.s, order = c(1, 0, 0), include.mean = F, method = "CSS-ML")

 Coefficients :
          ar1
       0.9250
s.e.   0.0423

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。

```
> me.arma11(arma11.s)
       ar1         ma1
 0.6377807 -0.2038076
> arima(arma11.s,c(1,0,1),include.mean = F)

Call:
arima(x = arma11.s, order = c(1, 0, 1), include.mean = F)

 Coefficients :
          ar1      ma1
       0.5877   0.3417
s.e.   0.1161   0.1579

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。