MODEL-BASED ESTIMATION FOR ARMA MODELS

For a zero mean ARMA(p,q) model, we can make most efficient estimation of the autocovariance and most accurate forecasting using *model-based* estimation provided we can correctly specify the model concerned. This is because the acvf is entirely specified by the $p+q+1$ parameters $\phi_1,\ldots,\phi_p,\theta_1,\ldots,\theta_q,\sigma_Z^2$, and typically $p+q+1$ is much smaller than n.

Yule-Walker Estimation: The Yule-Walker method can be applied to zero mean, stationary AR(p) processes;

$$X_t = \sum_{j=1}^{p} \phi_j X_{t-j} + Z_t.$$

We multiply both sides by X_{t-h} for $h = 1,\ldots,p$, and take expectations to get

$$\mathbb{E}[X_t X_{t-h}] = \sum_{j=1}^{p} \phi_j \mathbb{E}[X_{t-j}X_{t-h}] \qquad \Longrightarrow \qquad \gamma_X(h) = \sum_{j=1}^{p} \phi_j \gamma_X(h-j)$$

for $h = 1,\ldots,p$. Taking these p equations together, we obtain $\Gamma_X(p)\boldsymbol{\phi} = \gamma_X(p,1)$ or

$$\begin{bmatrix} \gamma_X(0) & \gamma_X(1) & \gamma_X(2) & \cdots & \gamma_X(p-1) \\ \gamma_X(1) & \gamma_X(0) & \gamma_X(1) & \cdots & \gamma_X(p-2) \\ \gamma_X(2) & \gamma_X(1) & \gamma_X(0) & \cdots & \gamma_X(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_X(p-1) & \gamma_X(p-2) & \gamma_X(p-3) & \cdots & \gamma_X(0) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_p \end{bmatrix} = \begin{bmatrix} \gamma_X(1) \\ \gamma_X(2) \\ \gamma_X(3) \\ \vdots \\ \gamma_X(p) \end{bmatrix}.$$

We may now substitute in the empirical estimates for the acv values in the matrix and in the vector on the right-hand side, and solve the resulting equation to obtain the estimates $\widehat{\boldsymbol{\phi}} = (\widehat{\phi}_1,\ldots,\widehat{\phi}_p)^\top$.

EXAMPLE: Consider the AR(4) model given by $\Phi(\mathrm{B})X_t = Z_t$, where polynomial $\Phi(\cdot)$ is given by

$$\Phi(z) = (1-0.5z)(1+0.4z)(1-0.2z)^2$$

so that the reciprocal roots are $0.5, -0.4, 0.2$ and 0.2. This means that the model is stationary and causal, and the coefficients are computed as follows

```
alpha.vals<-c(0.5,-0.4,0.2,0.2)
ar.coefs<-rep(1,length(alpha.vals)+1)
for(j in 1:length(alpha.vals)){
        ar.coefs.old<-ar.coefs
        if(j > 1){
                for(k in 2:j){
                        ar.coefs[k]<-ar.coefs.old[k]-alpha.vals[j]*ar.coefs.old[k-1]
                }
        }
        ar.coefs[j+1]<--alpha.vals[j]*ar.coefs.old[j]
}
(phi.vec<--ar.coefs[-1])   #These are the AR coefficients

+ [1]  0.500  0.120 -0.076  0.008
```

that is $\phi_1 = 0.5, \phi_2 = 0.12, \phi_3 = -0.076, \phi_4 = 0.008$. Numerically we have

```
p<-4
gammaX<-ARMAacf(ar=phi.vec,lag.max=p);gammaX

+          0          1          2          3          4
+ 1.00000000 0.53935940 0.35150039 0.16878820 0.09358283

gammaX.mat<-toeplitz(gammaX[-(p+1)])
solve(gammaX.mat,gammaX[-1])   #Check that the true phi values are a solution

+ [1]  0.500  0.120 -0.076  0.008
```

Now we repeat the calculation with a simulated data set, replacing $\Gamma_X(p)$ by $\widehat{\Gamma}_n(X)$.

```
n<-500
X<-arima.sim(n,model=list(ar=phi.vec))
gammaX.hat<-acf(X,plot=F,lag.max=p)$acf
gammaX.mat.hat<-toeplitz(gammaX.hat[-(p+1)])
solve(gammaX.mat.hat,gammaX.hat[-1])   #Estimates

+ [1]  0.47412711  0.11679759  0.05711220 -0.02856452
```

These estimates can be obtained using the `ar` function:

```
ar(X,order.max=4,aic=FALSE,method='yule-walker')

+
+ Call:
+ ar(x = X, aic = FALSE, order.max = 4, method = "yule-walker")
+
+ Coefficients:
+       1        2        3        4
+  0.4741   0.1168   0.0571  -0.0286
+
+ Order selected 4  sigma^2 estimated as  0.9176
```

Ordinary Least Squares: We can also obtain parameter estimates for an AR model by ordinary least squares (OLS) by minimizing

$$\sum_{t=p+1}^{n}\left(x_t-\sum_{j=1}^{p}\phi_j x_{t-j}\right)^2$$

which is a straightforward problem that can be solved analytically using calculus. In `R` to solve the OLS problem we may use the `lm` function. In the example above, if we want to assume that the process is zero mean, then we proceed as follows:

```
Y<-X[(p+1):n]
X1<-X[p:(n-1)]
X2<-X[(p-1):(n-2)]
X3<-X[(p-2):(n-3)]
X4<-X[(p-3):(n-4)]
summary(lm(Y~-1+X1+X2+X3+X4))

+
+ Call:
+ lm(formula = Y ~ -1 + X1 + X2 + X3 + X4)
+
+ Residuals:
+      Min       1Q   Median       3Q      Max
+ -2.63598 -0.69239 -0.03375  0.57007  2.77005
+
+ Coefficients:
+    Estimate Std. Error t value            Pr(>|t|)
+ X1  0.47521    0.04504  10.551 <0.0000000000000002 ***
+ X2  0.12199    0.04979   2.450              0.0146 *
+ X3  0.06113    0.04981   1.227              0.2203
+ X4 -0.02400    0.04491  -0.534              0.5933
+ ---
+ Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
+
+ Residual standard error: 0.9585 on 492 degrees of freedom
+ Multiple R-squared:  0.3329,Adjusted R-squared:  0.3274
+ F-statistic: 61.37 on 4 and 492 DF,  p-value: < 0.00000000000000022
```

The ar function can also be used to obtain the estimates.

```
ar(X,order.max=4,demean=FALSE,aic=FALSE,method='ols')

+
+ Call:
+ ar(x = X, aic = FALSE, order.max = 4, method = "ols", demean = FALSE)
+
+ Coefficients:
+       1        2        3        4
+  0.4752   0.1220   0.0611  -0.0240
+
+ Order selected 4  sigma^2 estimated as  0.9113
```

We can also proceed by estimating the mean and subtracting it prior to the estimation:

```
mu.hat<-mean(X)
mu.hat          #Estimated mean

+ [1] -0.1703939

Y<-X[(p+1):n]-mu.hat
X1<-X[p:(n-1)]-mu.hat
X2<-X[(p-1):(n-2)]-mu.hat
X3<-X[(p-2):(n-3)]-mu.hat
X4<-X[(p-3):(n-4)]-mu.hat
summary(lm(Y~-1+X1+X2+X3+X4))

+
+ Call:
+ lm(formula = Y ~ -1 + X1 + X2 + X3 + X4)
+
+ Residuals:
+      Min       1Q   Median       3Q      Max
+ -2.57810 -0.62216  0.02505  0.63193  2.80632
+
+ Coefficients:
+    Estimate Std. Error t value            Pr(>|t|)
+ X1  0.47047    0.04503  10.448 <0.0000000000000002 ***
+ X2  0.11937    0.04969   2.402              0.0167 *
+ X3  0.05868    0.04971   1.180              0.2384
+ X4 -0.02884    0.04490  -0.642              0.5211
+ ---
+ Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
+
+ Residual standard error: 0.9561 on 492 degrees of freedom
+ Multiple R-squared:  0.3207,Adjusted R-squared:  0.3152
+ F-statistic: 58.06 on 4 and 492 DF,  p-value: < 0.00000000000000022

ar(X-mu.hat,order.max=4,demean=FALSE,aic=FALSE,method='ols')

+
+ Call:
+ ar(x = X - mu.hat, aic = FALSE, order.max = 4, method = "ols",     demean = FALSE)
+
+ Coefficients:
+       1        2        3        4
+  0.4705   0.1194   0.0587  -0.0288
+
+ Order selected 4  sigma^2 estimated as  0.9067
```

Burg's method: Burg's method for an AR process uses the partial autocorrelations to lag p. Using the `ar` function:

```
ar(X,order.max=4,demean=FALSE,aic=FALSE,method='burg')

+
+ Call:
+ ar(x = X, aic = FALSE, order.max = 4, method = "burg", demean = FALSE)
+
+ Coefficients:
+       1        2        3        4
+  0.4789   0.1195   0.0588  -0.0241
+
+ Order selected 4  sigma^2 estimated as  0.912
```

Maximum likelihood: A maximum likelihood approach requires a further assumption that specifies the conditional probability distribution of each of the X_t. Usually it is assumed that $Z_t \sim Normal(0, \sigma_Z^2)$, so that for $t > p$

$$X_t | X_1, \ldots, X_{t-1} \sim Normal(\phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p}, \sigma_Z^2)$$

from which we can construct the likelihood function in the usual way. For $t = 1, \ldots, p$, we may deduce the distribution of X_t using the stationarity of the process, and deduce that

$$X_{1:n} \sim Normal_n(\mathbf{0}, \Gamma_X(n))$$

Here $\Gamma_X(n)$ depends on $\boldsymbol{\phi}$, so we write $\Gamma_X(n; \boldsymbol{\phi})$, so that the full likelihood for data $\mathbf{x}_{1:n}$ is given by

$$\mathcal{L}_n(\boldsymbol{\phi}) = f(x_1, \ldots, x_p; \boldsymbol{\phi}) \prod_{t=p+1}^{n} f(x_t | x_1, \ldots, x_{t-1}; \boldsymbol{\phi}). \tag{1}$$

Up to a constant the corresponding log-likelihood is

$$\ell_n(\boldsymbol{\phi}) = -\frac{1}{2} \log |\Gamma_X(n; \boldsymbol{\phi})| - \frac{1}{2} \mathbf{x}_{1:n}^\top \{\Gamma_X(n; \boldsymbol{\phi})\}^{-1} \mathbf{x}_{1:n}.$$

To maximize this function, we need to compute the matrix inverse and the determinant. This can be achieved efficiently using the Levinson-Durbin or Innovations algorithms, or using eigenvalue/singular value decompositions where we write

$$\Gamma_X(n; \boldsymbol{\phi}) = \mathbf{U}\mathbf{D}\mathbf{U}^\top \qquad \therefore \qquad \{\Gamma_X(n; \boldsymbol{\phi})\}^{-1} = \mathbf{U}\mathbf{D}^{-1}\mathbf{U}^\top$$

with $\mathbf{D}$ a diagonal matrix with elements $d_1, \ldots, d_n$, and

$$|\Gamma_X(n; \boldsymbol{\phi})| \equiv \det \Gamma_X(n; \boldsymbol{\phi}) = \prod_{t=1}^{n} d_t$$

```
ar(X,order.max=4,demean=FALSE,aic=FALSE,method='mle')

+
+ Call:
+ ar(x = X, aic = FALSE, order.max = 4, method = "mle", demean = FALSE)
+
+ Coefficients:
+       1        2        3        4
+  0.4779   0.1187   0.0594  -0.0232
+
+ Order selected 4  sigma^2 estimated as  0.9119
```

The maximum likelihood method can be used for ARMA(p, q) processes.

EXAMPLE: Consider the AR$(2, 3)$ model

$$X_t = 0.1X_{t-1} - 0.56X_{t-2} + Z_t + 0.2Z_{t-1} - 0.53Z_{t-2} + 0.09Z_{t-3}.$$

In this model, we have

$$\Phi(z) = 1 - 0.1z - 0.56z^2 = (1 - 0.8z)(1 + 0.7z)$$

so the model is stationary and causal, and

$$\Theta(z) = (1 - 0.2z)(1 - 0.5z)(1 + 0.9z)$$

so the model is invertible. The $\Theta(z)$ coefficients are computed as

```
beta.vals<-c(0.2,0.5,-0.9)
ma.coefs<-rep(1,length(beta.vals)+1)
for(j in 1:length(beta.vals)){
        ma.coefs.old<-ma.coefs
        if(j > 1){
                for(k in 2:j){
                        ma.coefs[k]<-ma.coefs.old[k]-beta.vals[j]*ma.coefs.old[k-1]
                }
        }
        ma.coefs[j+1]<--beta.vals[j]*ma.coefs.old[j]
}
(theta.vec<-ma.coefs[-1])

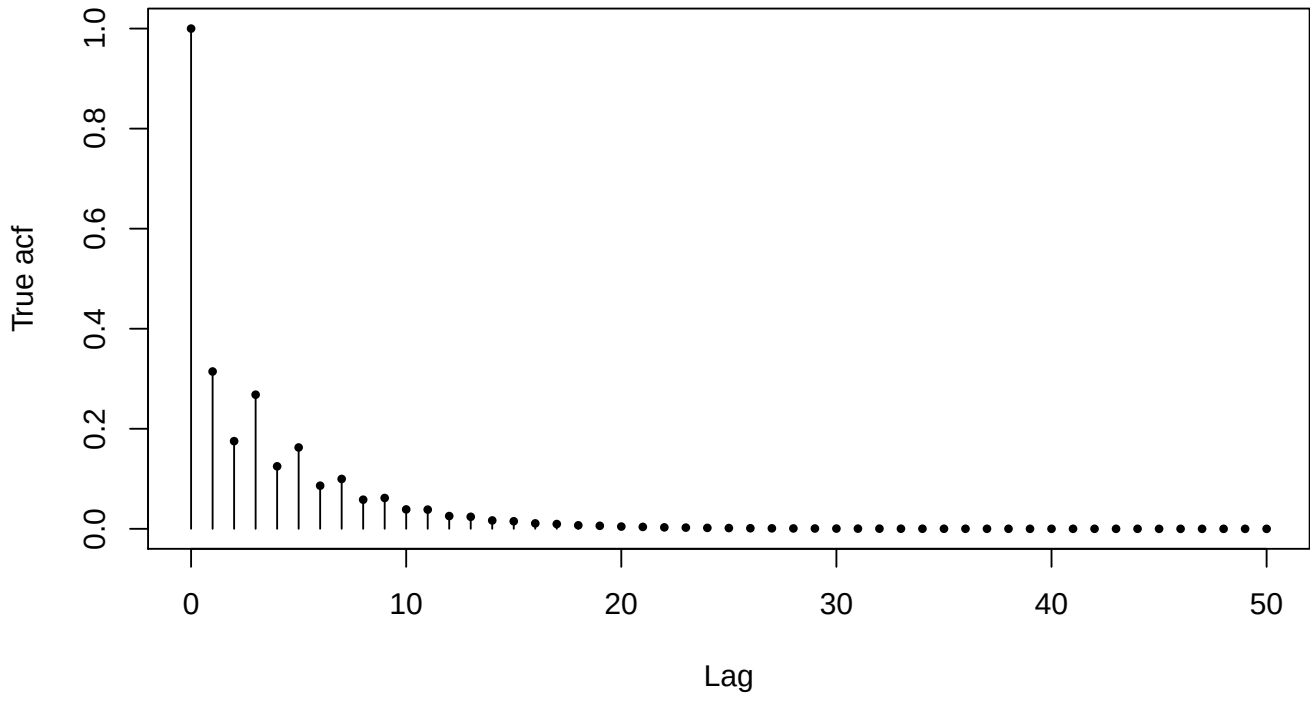
+ [1]  0.20 -0.53  0.09
```

The acf of this model shows slow decay

```
phi.vec<-c(0.1,0.56);true.pars<-c(phi.vec,theta.vec)
acfs<-ARMAacf(ar=phi.vec,ma=theta.vec,lag.max=50)
acfs[1:10]

+          0          1          2          3          4          5
+ 1.00000000 0.31452639 0.17526769 0.26812804 0.12496271 0.16264797
+          6          7          8          9
+ 0.08624392 0.09970726 0.05826732 0.06166280

par(mar=c(4,4,1,0))
plot(0:50,acfs,pch=19,cex=0.5,xlab='Lag',ylab='True acf')
for(i in 0:50){lines(c(i,i),c(0,acfs[1+i]))}
```



We start with a very large sample of $n = 2000$:

```
n<-2000
set.seed(333)
X<-arima.sim(n,model=list(ar=phi.vec,ma=theta.vec))
(ml1<-arima(X,order=c(2,0,3),method='ML',include.mean = FALSE))

+
+ Call:
+ arima(x = X, order = c(2, 0, 3), include.mean = FALSE, method = "ML")
+
+ Coefficients:
+          ar1     ar2     ma1      ma2     ma3
+       0.1468  0.5169  0.1635  -0.5078  0.1262
+ s.e.  0.0485  0.0508  0.0509   0.0565  0.0349
+
+ sigma^2 estimated as 0.9815:  log likelihood = -2819.49,  aic = 5650.97
```

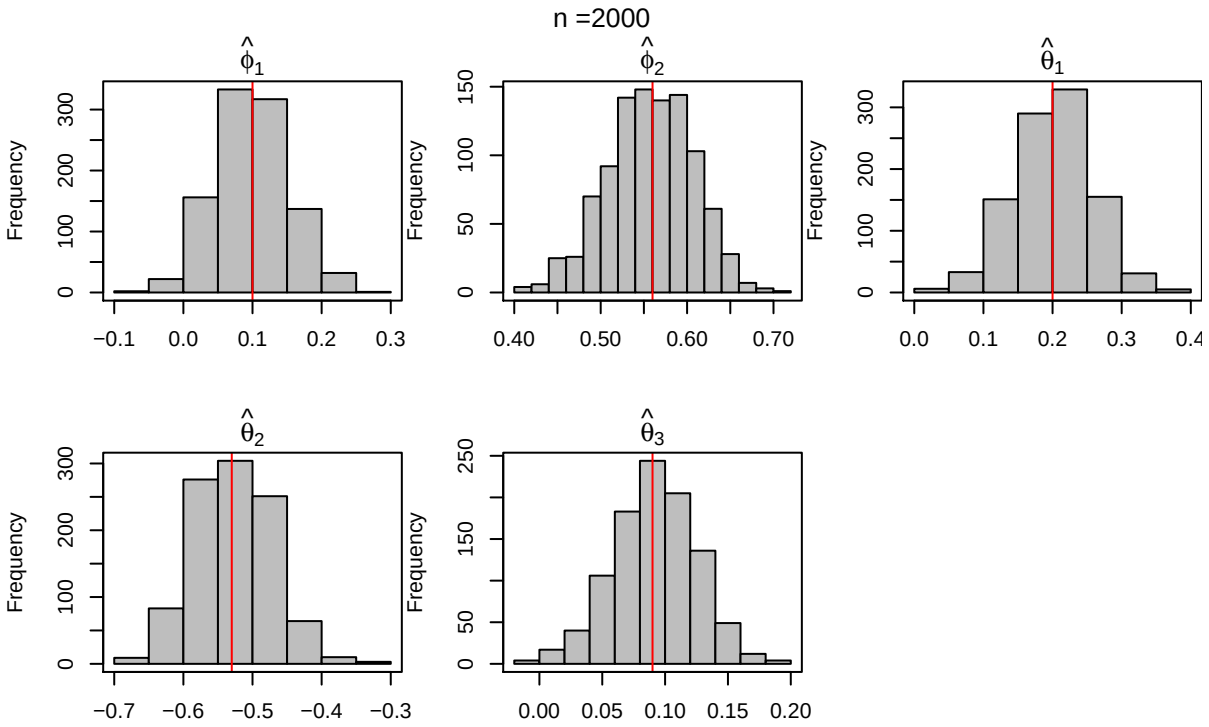
The estimation procedure recovers the true values of the parameters quite accurately. For a smaller sample size, the estimation is more inaccurate.

```
n<-200
set.seed(333)
X<-arima.sim(n,model=list(ar=phi.vec,ma=theta.vec))
(ml2<-arima(X,order=c(2,0,3),method='ML',include.mean = FALSE))

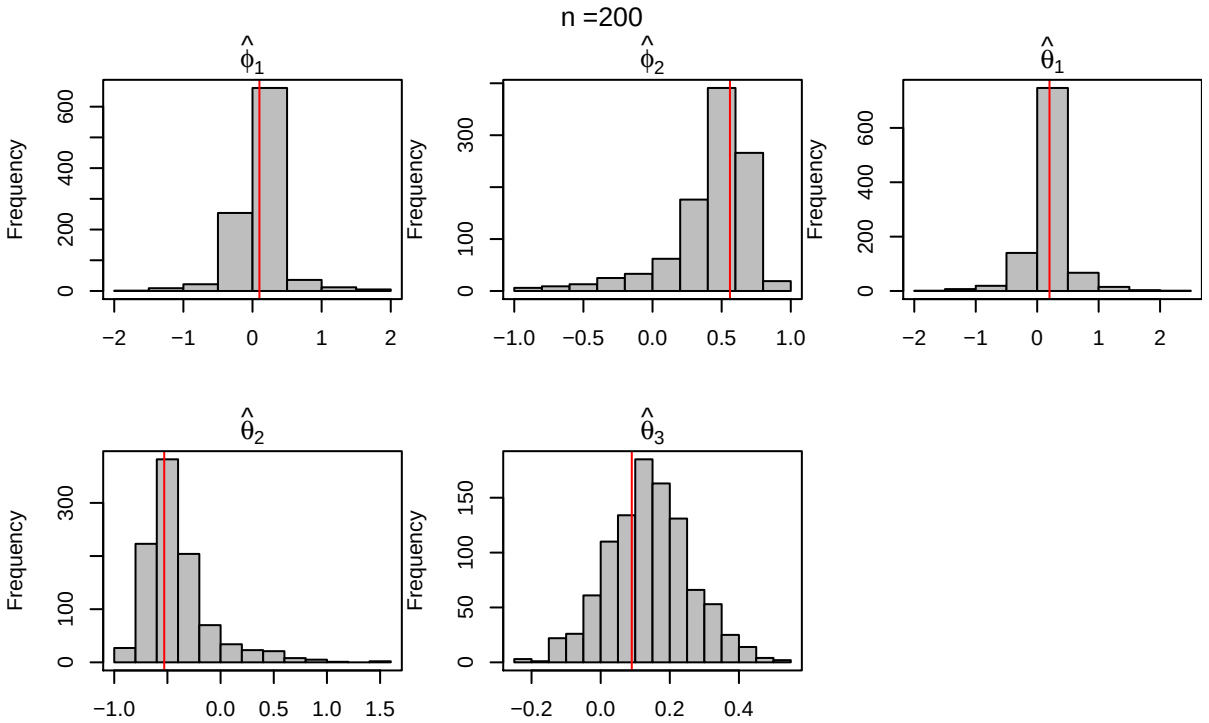
+
+ Call:
+ arima(x = X, order = c(2, 0, 3), include.mean = FALSE, method = "ML")
+
+ Coefficients:
+          ar1      ar2      ma1      ma2     ma3
+       0.6612  -0.1125  -0.2010  -0.0960  0.3342
+ s.e.  0.2991   0.2578   0.2881   0.1464  0.1015
+
+ sigma^2 estimated as 0.8763:  log likelihood = -270.88,  aic = 553.76
```

We can run a simulation study to assess the accuracy of inference for the different sample size, and run 1000 replications. The following code runs the simulation and stores the parameter estimates.

```
n1<-2000;n2<-200;nreps<-1000
ests.mat1<-ests.mat2<-matrix(0,nrow=nreps,ncol=5)
oldw <- getOption("warn")
options(warn=-1)
for(irep in 1:nreps){
    X<-arima.sim(n1,model=list(ar=phi.vec,ma=theta.vec))
    ml1<-arima(X,order=c(2,0,3),method='ML',include.mean = FALSE)
    ests.mat1[irep,]<-coef(ml1)
    X<-arima.sim(n2,model=list(ar=phi.vec,ma=theta.vec))
    ml2<-arima(X,order=c(2,0,3),method='ML',include.mean = FALSE)
    ests.mat2[irep,]<-coef(ml2)
}
options(warn=oldw)
par(mfrow=c(2,3),mar=c(4,4,2,0),oma=c(0,0,1,0))
tvec<-c(expression(hat(phi)[1]),expression(hat(phi)[2]),
        expression(hat(theta)[1]),expression(hat(theta)[2]),expression(hat(theta)[3]))
for(i in 1:5){
    hist(ests.mat1[,i],col='gray',main=tvec[i],xlab=' ');abline(v=true.pars[i],col='red')
    box()
}
title(substitute(paste('n =',nval),list(nval = n1)), outer=TRUE)
par(mfrow=c(2,3),mar=c(4,4,2,0),oma=c(0,0,1,0))
```




```
for(i in 1:5){
    hist(ests.mat2[,i],col='gray',main=tvec[i],xlab=' ');abline(v=true.pars[i],col='red')
    box()
}
title(substitute(paste('n =',nval),list(nval = n2)), outer=TRUE)
```



The red lines mark the positions of the true values:

Note: In all of the inference methods, it is the information in the autocovariances that determines quality of the parameter estimation. As different ARMA models can have almost identical acvs, it may be difficult in general to identify particular parameters.

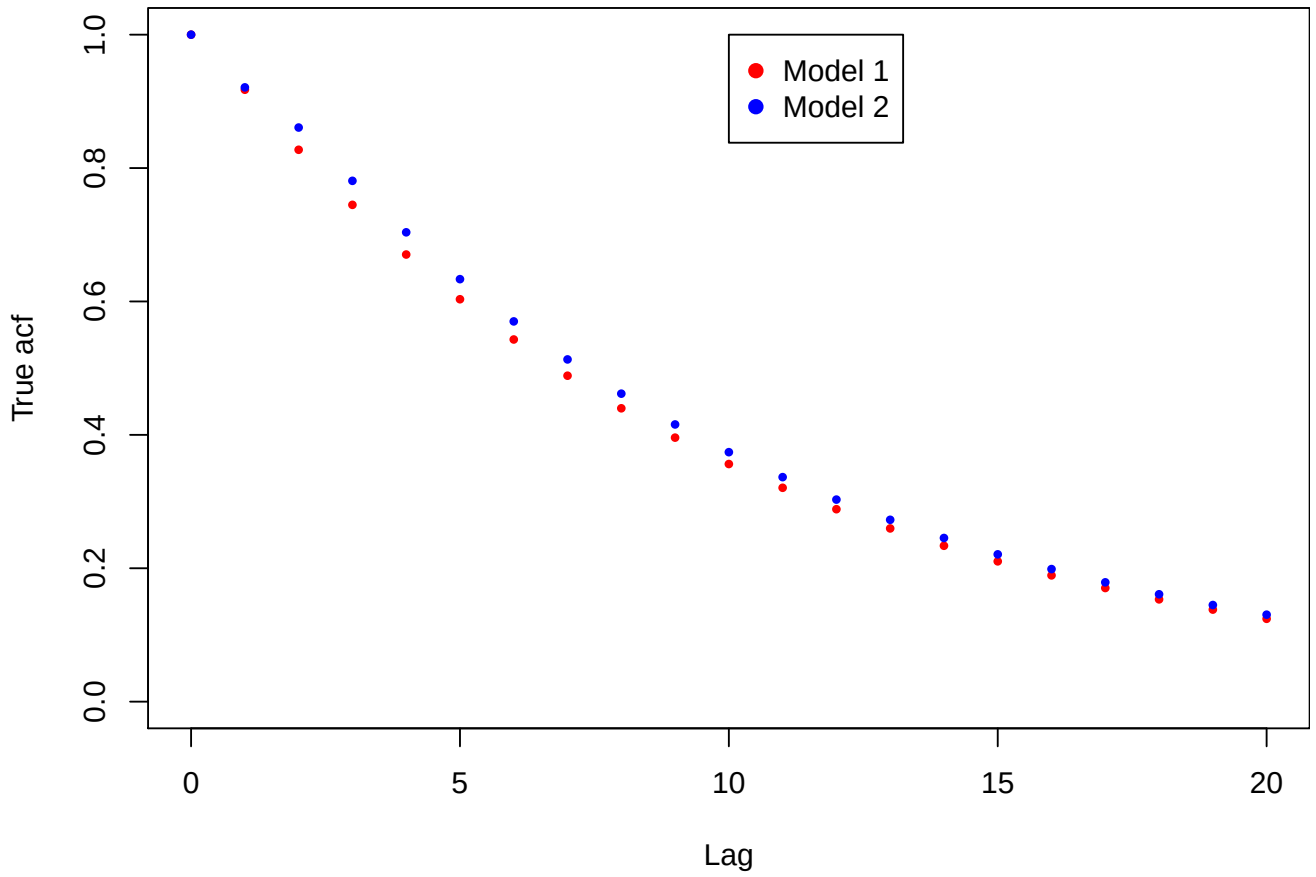
EXAMPLE: Consider the AR(2) model

$$X_t = X_{t-1} - 0.09X_{t-2} + Z_t$$

and the ARMA(3, 2) model

$$X_t = 1.1X_{t-1} - 0.19X_{t-2} + 0.009X_{t-3} + Z_t - 0.25Z_{t-1} + 0.2Z_{t-2}$$

```
phi.vec<-c(1,-0.09)
acf.1<-ARMAacf(ar=phi.vec,lag.max=20)
phi.vec<-c(1.1,-0.19,0.009);theta.vec<-c(-0.25,0.2)
acf.2<-ARMAacf(ar=phi.vec,ma=theta.vec,lag.max=20)
par(mar=c(4,4,1,0))
plot(0:20,acf.1,pch=19,cex=0.5,xlab='Lag',ylab='True acf',col='red',ylim=range(0,1))
points(0:20,acf.2,pch=19,cex=0.5,col='blue')
legend(10,1.0,c("Model 1","Model 2"),pch=19,col=c('red','blue'))
```



Clearly the acf values are almost identical.