Time Series: A First Course with Bootstrap Starter

Contents

Lesson 10-1: MA Identification

- We can test for truncation of the sample ACVF to identify the order of an MA model.

Paradigm 10.1.2. Sequential Testing for the MA Order

- To identify an MA model, we must use the sample ACF rather than the true ACF.
- Take as null hypothesis that the process is an MA(q), such that

$$H_0 : \gamma(k) = 0 \quad \text{for all } k > q.$$

- We can use the estimate $\widehat{\rho}(k)$ as test statistic, for any $k > q$.
- Under H_0, the asymptotic variance (using Bartlett's Formula, Remark 9.6.10) is n^{-1} times

$$\frac{\langle f^2 \rangle_0}{\langle f \rangle_0^2} = \sum_{|h| \leq q} \rho(h)^2.$$

- Plugging in sample estimates, we obtain the test statistic

$$\sqrt{n} \frac{\widehat{\rho}(k)}{\sqrt{\sum_{|h| \leq q} \widehat{\rho}(h)^2}}$$

  for each $k > q$.
- How to use this: choose a q to test, and compute the test statistics for various $k > q$. If they fail to reject, keep that value of q; otherwise, increment q and repeat the procedure.
- However, this procedure does not account for multiple testing!

Paradigm 10.1.4. Joint Testing for the MA Order

- We want to test at multiple lags.
- H_0 is equivalent to $0 = \max\{|\rho(k)|, k > q\}$.

Empirical Rule

- Let $\widehat{q}$ be the smallest positive integer such that

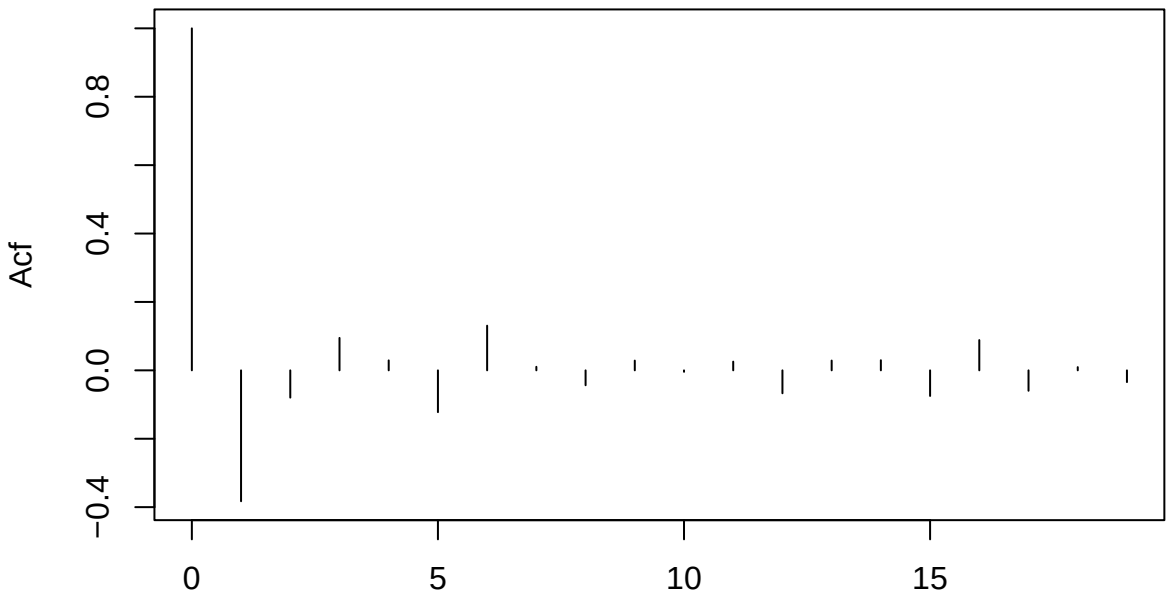
$$|\widehat{\rho}(\widehat{q} + k)| < c\sqrt{n^{-1} \log n}$$

  for all $k = 1, \ldots, K_n$, where $c > 0$ is a fixed constant, and K_n is a positive, non-decreasing integer-valued function of n such that $K_n = o(\log n)$.
- $\widehat{q}$ is consistent for q.
- Choose $c = 1.96$ and $K_n = 1 + [3\sqrt{\log n}]$

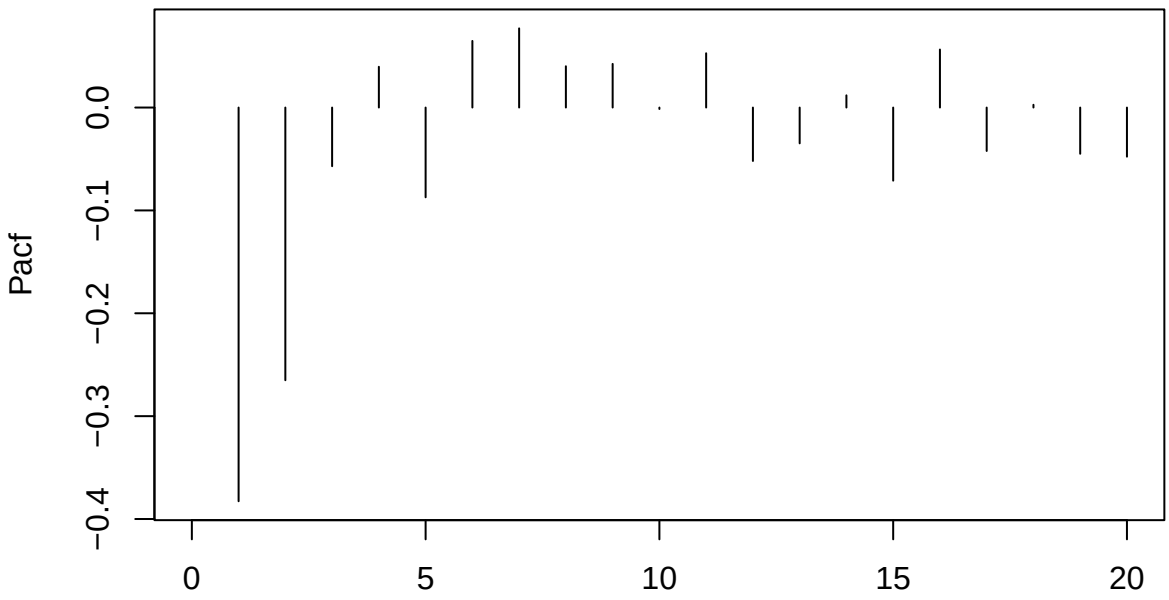
Example 10.1.5. Joint Testing Identification of Non-Defense Capitalization

- We now apply the Empirical Rule to the Non-defense capitalization data.
- We take first differences because of non-stationarity.
- The first 20 sample autocorrelations and partial autocorrelations are plotted.

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
ndc.diff <- diff(ndc)
n <- length(ndc.diff)
gamma.hat <- acf(ndc.diff,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
kappa.hat <- pacf(ndc.diff,lag=n-1,plot=FALSE)$acf[,,1]
plot(ts(gamma.hat[1:20]/gamma.hat[1],start=0),xlab="",ylab="Acf",type="h")
```




```
plot(ts(c(NA,kappa.hat[1:20]),start=0),xlab="",ylab="Pacf",type="h")
```



- From visual inspection, we might guess that $q = 1$.
- Then we apply the Empirical Rule, where $n = 292$ implies $K_n = 5$.
- We print K_n values of the sample autocorrelations.

```
alpha <- .05
crit <- qnorm(1-alpha/2)
K.n <- 1 + floor(3*sqrt(log(n,base=10)))
rho.hat <- gamma.hat[-1]/gamma.hat[1]
print(rho.hat[1:K.n])
```

```
## [1] -0.38271473 -0.07977748  0.09482173  0.02917913 -0.12226110
```

```
rho.test <- rho.hat/sqrt(log(n,base=10)/n)
k <- 1
while(k < (n-K.n))
{
    if(max(abs(rho.test[k:(k+K.n-1)])) < crit)
    {
        q.hat <- k-1
        k <- n-K.n
    } else { k <- k+1 }
}
```

- The selected model order is 1, which agrees with the ACF and PACF plots.

Lesson 10-3: AR Identification

- We can determine AR model order by examining truncation of the PACF.

Remark 10.3.1. AR Model Identification

- To identify an AR model, we must use the sample PACF rather than the true PACF.
- Take as null hypothesis that the process is an AR(p), such that

$$H_0 : \kappa(k) = 0 \quad \text{for all } k > p.$$

- We can use the estimate $\widehat{\kappa}(k)$ as test statistic, for any $k > p$.
- We require an asymptotic distribution theory.

Corollary 10.3.4.

- Suppose $\{X_t\}$ is an AR(p) process with mean μ and i.i.d. inputs of variance σ^2. Then

$$\sqrt{n}(\widehat{\kappa}(k) - \kappa(k)) \Rightarrow \mathcal{N}(0, \sigma^2 \underline{e}_k' \Gamma_k^{-1} \underline{e}_k),$$

  where $\underline{e}_k$ is the unit vector with one in last (index k) position.
- So the asymptotic variance is the bottom right entry of Γ_k^{-1} times σ^2.

Paradigm 10.3.6. AR Model Identification

- Suppose the true model is AR(p), and we test H_0 with $k > p$.
- Trick: the process is also an AR(k), where the last $k - p$ coefficients are zero!
- So we can apply Corollary 10.3.4, and it can be shown that because $k > p$ the limiting variance is 1, i.e.,

$$\sqrt{n}\,\widehat{\kappa}(k) \Rightarrow \mathcal{N}(0, 1)$$

  under H_0.

- So we can do sequential testing using the sample PACF, with the same cautions about multiple testing.
- We also have an *Empirical Rule* for AR identification.

Empirical Rule

- Let $\widehat{p}$ be the smallest positive integer such that

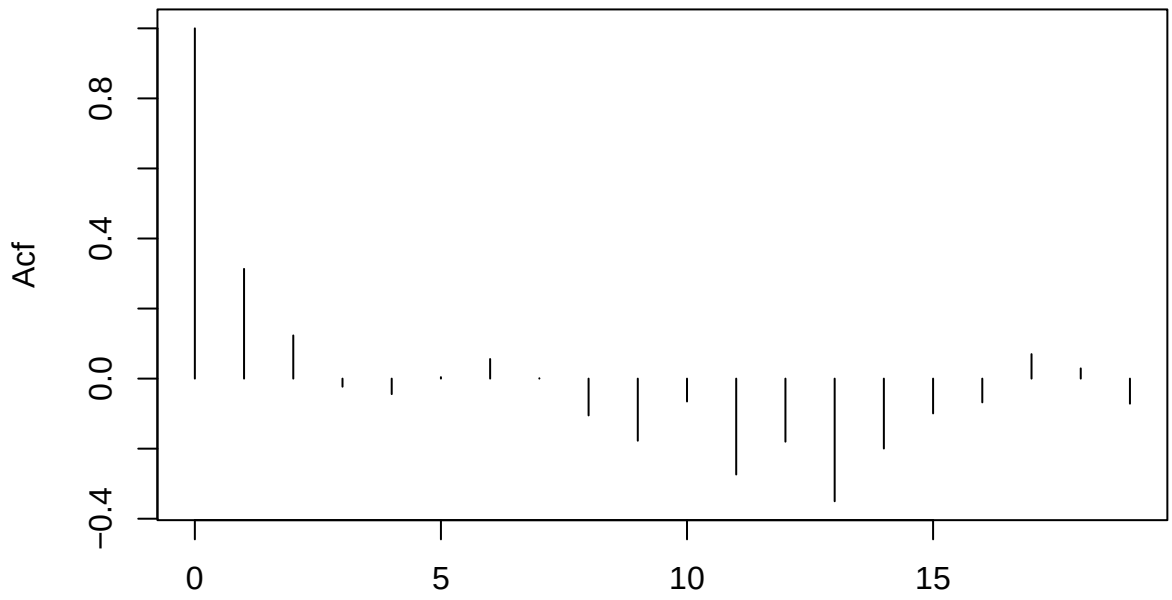
$$|\widehat{\kappa}(\widehat{p} + k)| < c\sqrt{n^{-1} \log n}$$

  for all $k = 1, \ldots, K_n$, where $c > 0$ is a fixed constant, and K_n is a positive, non-decreasing integer-valued function of n such that $K_n = o(\log n)$.
- Choose $c = 1.96$ and $K_n = 1 + [3\sqrt{\log n}]$

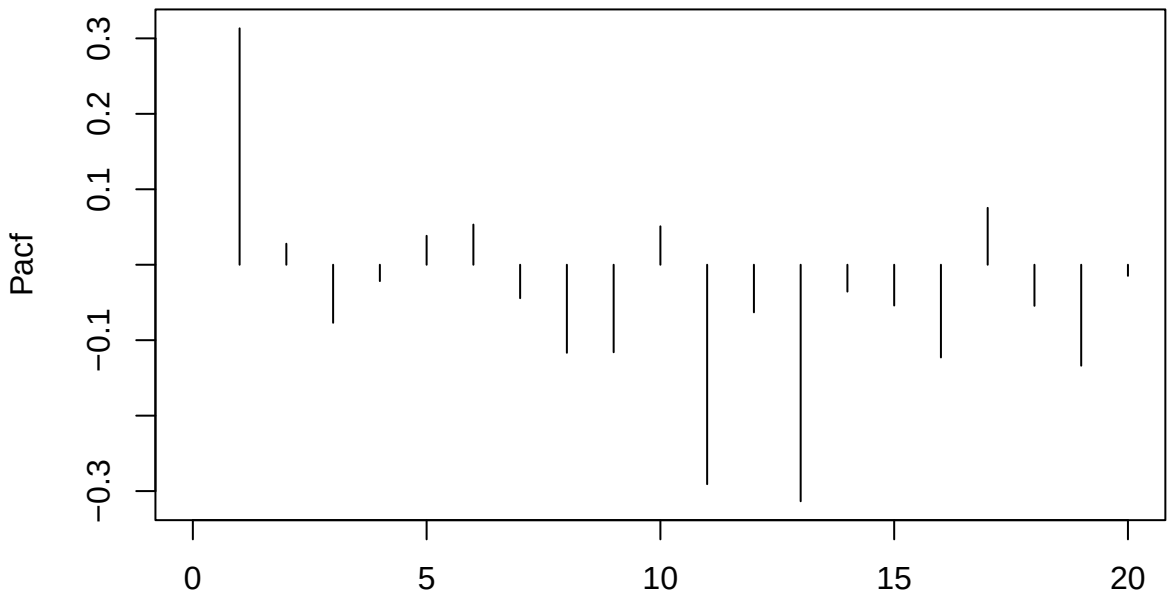
Example 10.3.7. Identification of Urban World Population

- We now apply the Empirical Rule to the Urban World Population time series.
- We take second differences because of non-stationarity.
- The first 20 sample autocorrelations and partial autocorrelations are plotted.

```
urban <- read.table("urbanpop.dat")
urban <- ts(urban[67:1,], start = 1951)
urban.diff <- diff(diff(urban))
n <- length(urban.diff)
gamma.hat <- acf(urban.diff,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
kappa.hat <- pacf(urban.diff,lag=n-1,plot=FALSE)$acf[,,1]
plot(ts(gamma.hat[1:20]/gamma.hat[1],start=0),xlab="",ylab="Acf",type="h")
```




```
plot(ts(c(NA,kappa.hat[1:20]),start=0),xlab="",ylab="Pacf",type="h")
```



- From visual inspection, we might guess that $p = 0$, thought there seems to be non-trivial correlation at lags 11 and 13.
- Then we apply the Empirical Rule, where $n = 65$ implies $K_n = 5$.
- We print K_n values of the sample autocorrelations, and $\widehat{p}$.

```
alpha <- .05
crit <- qnorm(1-alpha/2)
K.n <- 1 + floor(3*sqrt(log(n,base=10)))
print(kappa.hat[1:K.n])
```

```
## [1]  0.31328846  0.02786067 -0.07692737 -0.02165771  0.03832985
```

```
kappa.test <- kappa.hat/sqrt(log(n,base=10)/n)
k <- 1
while(k < (n-K.n))
{
    if(max(abs(kappa.test[k:(k+K.n-1)])) < crit)
    {
        p.hat <- k-1
        k <- n-K.n
    } else { k <- k+1 }
}
```

- This results in AR identification of order 0.

Lesson 10-4: Optimal Prediction Estimators

- We provide some initial results for fitting models, based on optimal prediction.

Fact 10.4.1. Asymptotic Prediction Error Variance

- Suppose $\{X_t\}$ is a linear time series with spectral density f.
- Consider the 1-step ahead forecast based on moving average representation $\psi(z) = \sum_{k\geq 0} \psi_k z^k$. The predictor (by Paradigm 7.4.3) is

$$\widehat{X}_{n+1} = \sum_{k\geq 0} \psi_{k+1} B^k \, \psi(B)^{-1} X_n.$$

- Letting $g(\lambda) = |\psi(e^{-i\lambda})|^{-2}$, the *prediction error variance* is

$$\mathcal{V}_\infty = \frac{1}{2\pi} \int_{-\pi}^{\pi} g(\lambda) f(\lambda) \, d\lambda = \langle g \, f \rangle_0.$$

- This is estimated via

$$\widehat{\mathcal{V}_\infty} = \langle g \, I \rangle_0.$$

- Note that we need not have $f(\lambda) = \sigma^2/g(\lambda)$, because the model ($\psi(z)$) might differ from the true process (f).
- We can fit models by finding $\psi(z)$ so as to minimize $\widehat{\mathcal{V}_\infty}$.

Paradigm 10.4.2. Fitting an AR(p) Model

- Suppose the model is an AR(p), so that $\psi(z) = (1 - \sum_{j=1}^{p} \phi_j z^j)^{-1}$ and

$$g(\lambda) = |1 - \sum_{j=1}^{p} \phi_j e^{-i\lambda j}|^2.$$

- The prediction error variance is

$$\mathcal{V}_\infty = \gamma(0) - 2\underline{\phi}' \underline{\gamma}_p + \underline{\phi}' \, \Gamma_p \, \underline{\phi},$$

  where $\underline{\phi}' = [\phi_1, \ldots, \phi_p]$.
- The estimated prediction error variance is

$$\widehat{\mathcal{V}_\infty} = \widehat{\gamma}(0) - 2\underline{\phi}' \widehat{\underline{\gamma}}_p + \underline{\phi}' \, \widehat{\Gamma}_p \, \underline{\phi},$$

  which we can minimize with respect to $\underline{\phi}$. This yields the Yule-Walker estimator:

$$\widehat{\underline{\phi}} = \widehat{\Gamma}_p^{-1} \, \widehat{\underline{\gamma}}_p.$$

Paradigm 10.4.3. Fitting an MA(q) Model

- Suppose the model is an MA(q), so that $\psi(z) = 1 + \sum_{j=1}^{q} \theta_j z^j$ and

$$g(\lambda) = |1 + \sum_{j=1}^{q} \theta_j z^j|^{-2}.$$

- The prediction error variance is a complicated function of the MA coefficients. We can minimize the estimated prediction error variance to fit the model.

Example 10.4.6. Fitting an AR(p) to Wolfer Sunspots

- Consider the Wolfer sunspot time series.
- Applying Paradigm 10.3.6, we arrive at $\widehat{p} = 4$ by the Empirical Rule.
- We fit this AR(4) model using Paradigm 10.4.2.

```
wolfer <- read.table("wolfer.dat")
wolfer <- ts(wolfer,start=1749,frequency=12)
n <- length(wolfer)
gamma.hat <- acf(wolfer,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
p.order <- 4
phi.ar <- solve(toeplitz(gamma.hat[1:p.order])) %*% gamma.hat[2:(p.order+1)]
print(phi.ar)
```

```
##           [,1]
## [1,] 0.5937912
## [2,] 0.1258125
## [3,] 0.1049469
## [4,] 0.1354815
```

Lesson 10-5: Relative Entropy Minimization

- We can fit AR(p) models using the sample Yule-Walker equations (Paradigm 10.4.2), or using OLS.
- We can fit MA(q) models using a *spectral factorization*, which is akin to Yule-Walker in that it is a method-of-moments approach.
- Here we focus on the Whittle likelihood and the Gaussian likelihood.

Remark 10.5.1. Connection to Relative Entropy

- Minimizing asymptotic prediction variance is a method for fitting models.
- If the model has parameters $\underline{\omega}$, then
$$\widehat{\mathcal{V}}_\infty(\underline{\omega}) = \langle g_{\underline{\omega}}\, I \rangle_0$$
can be computed for any $\underline{\omega}$, and its minimizer (when unique) is the parameter estimate $\widehat{\underline{\omega}}$.
- Connection to relative entropy: let $\sigma^2 g_{\underline{\omega}}^{-1}$ be the model spectral density. Then
$$\log \sigma^2 + \langle \frac{I}{\sigma^2 g_{\underline{\omega}}^{-1}} \rangle_0 = \log \sigma^2 + \sigma^{-2}\, \widehat{\mathcal{V}}_\infty(\underline{\omega})$$
is the Kullback-Leibler (KL) discrepancy between periodogram and model spectrum. This is called the *Whittle likelihood*.
- KL is minimized with $\widehat{\underline{\omega}}$ and $\widehat{\sigma^2} = \widehat{\mathcal{V}}_\infty(\widehat{\underline{\omega}})$.
- We call $\widehat{\mathcal{V}}_\infty(\underline{\omega})$ the *Profile Whittle likelihood*.

Paradigm 10.5.9. The Gaussian Log Likelihood

- We consider the log likelihood of a length n sample from a stationary Gaussian time series. Multiplying by -2, we obtain the *Gaussian divergence*:
$$\mathcal{L}(\underline{\omega}, \sigma^2) = n \log(2\pi) + \log\det \Gamma_n + (\underline{X} - \underline{\mu})' \, \Gamma_n^{-1} \, (\underline{X} - \underline{\mu}).$$
- Here $\underline{\mu}$ is the mean vector, and equals the mean μ times a vector of ones.
- And $\underline{\omega}$ is the parameter vector (all the parameters except the input variance σ^2).
- Note that for AR, MA, and ARMA models, there is a parameter vector $\underline{\omega}$, which together with σ^2 determines $\gamma(h)$, and thereby Γ_n.
- To fit a model, we seek to minimize the Gaussian divergence over $\underline{\omega}$ and σ^2. The resulting minimizers are the Maximum Likelihood Estimators (MLEs) $\widehat{\underline{\omega}}$ and $\widehat{\sigma^2}$.

Remark 10.5.10. Profile Gaussian Log Likelihood

- For many models, we can factor out σ^2 from each $\gamma(h)$.
- In such a case, write $v(h) = \gamma(h)/\sigma^2$, and construct the Toeplitz Υ_n from these $v(h)$ values.
- Then the divergence becomes

$$\mathcal{L}(\underline{\omega}, \sigma^2) = n\log(2\pi) + n\log\sigma^2 + \log\det\Upsilon_n + \sigma^{-2}(\underline{X} - \underline{\mu})'\,\Upsilon_n^{-1}\,(\underline{X} - \underline{\mu}).$$

- Use calculus: differentiate with respect to σ^2, set equal to zero, and solve.
- Get a minimizer $\widehat{\sigma^2}$ that is a function of the still unknown $\underline{\omega}$.
- Plugging this formula back in to the divergence is called *concentration*, and we get a *profile likelihood* $\mathcal{L}(\underline{\omega}, \widehat{\sigma^2})$.
- The result is

$$\widehat{\sigma^2} = n^{-1}(\underline{X} - \underline{\mu})'\,\Upsilon_n^{-1}\,(\underline{X} - \underline{\mu})$$
$$\mathcal{L}(\underline{\omega}, \widehat{\sigma^2}) = n\log(2\pi) + (n+1)\log\widehat{\sigma^2} + \log\det\Upsilon_n.$$

- So we can minimize the profile likelihood as a function of $\underline{\omega}$, obtaining $\widehat{\underline{\omega}}$, and then finally get the input variance MLE via plugging into the first formula.
- For large n, it can be shown that $n^{-1}\log\det\Upsilon_n \approx 0$, which implies we can dump the last term in the profile likelihood. Then approximately, we can just minimize $\widehat{\sigma^2}$, which is interpretable as a measure of one-step ahead forecast mean squared error.

Exercise 10.34. Computing the Whittle Likelihood for MA Processes

- We write R code to compute the Whittle likelihood for an MA process.
- So $\underline{\omega} = [\theta_1, \ldots, \theta_q]'$, and $g(\lambda) = |1 + \sum_{j=1}^{q}\theta_j z^j|^{-2}$ (see Paradigm 10.4.3).
- First we write a function to compute the periodogram at the Fourier frequencies.

```
pgram.dft <- function(data)
{
    n <- length(data)
    gamma.hat <- acf(data,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
    lambda <- seq(-floor(n/2)+1,floor(n/2))*2*pi/n
    pgram <- cos(0*lambda)*gamma.hat[1]
    for(h in 1:(n-1))
    {
        pgram <- pgram + 2*cos(h*lambda)*gamma.hat[h+1]
    }
    pgram <- ts(pgram,start=0,frequency=n)
    return(pgram)
}
```

- Next, we encode the Whittle likelihood for the MA(q) model.

```
whittle.ma <- function(theta,sigma,pgram)
{
    n <- length(pgram)
    lambda <- seq(-floor(n/2)+1,floor(n/2))*2*pi/n
    q <- length(theta)
    f.spec <- rep(1,n)
    for(k in 1:q) { f.spec <- f.spec + theta[k]*exp(-1*1i*k*lambda) }
    f.spec <- Mod(f.spec)^2
    lik <- mean(pgram/f.spec)/sigma^2 + log(sigma^2)
    return(lik)
}
```

- Finally, we test this on the Non-Defense Capitalization data.
- We evaluate the Whittle likelihood for the MA(1) model with $\theta_1 = -.466$ and $\sigma^2 = .0053$ (these estimates are based on the spectral factorization method).

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
pgram <- pgram.dft(diff(ndc))
whittle.ma(-.466,sqrt(.0053),pgram) + log(2*pi)
```

```
## [1] -2.426116
```

Exercise 10.42. Profile Whittle Likelihood

- We extend Exercise 10.34 to code up the profile Whittle likelihood.

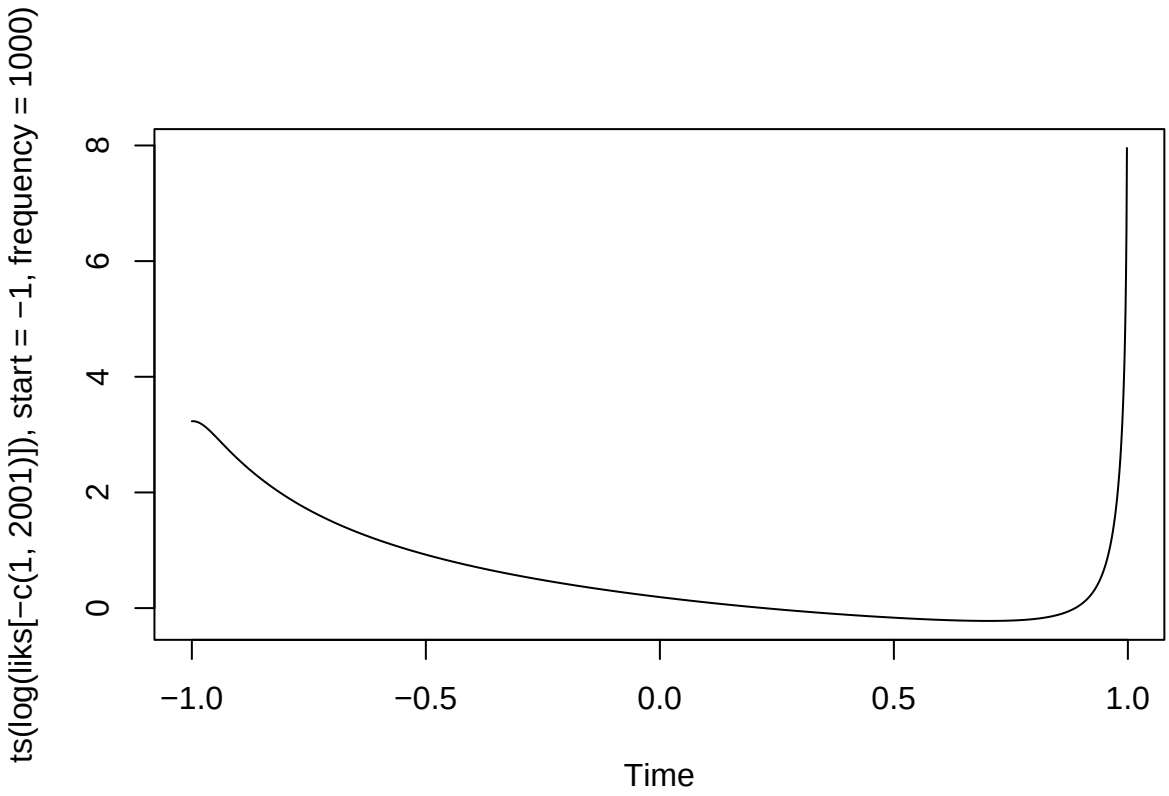
```
whittleprof.ma <- function(theta,pgram)
{
    n <- length(pgram)
    lambda <- seq(-floor(n/2)+1,floor(n/2))*2*pi/n
    q <- length(theta)
    f.spec <- rep(1,n)
    for(k in 1:q) { f.spec <- f.spec + theta[k]*exp(-1*1i*k*lambda) }
    f.spec <- Mod(f.spec)^2
    lik <- mean(pgram/f.spec)
    return(lik)
}
```

- We test this on a MA(1) simulation with $\theta_1 = .7$, $\sigma^2 = 1$, and $n = 100$.

```
armapq.sim <- function(n,burn,ar.coefs,ma.coefs,innovar)
{
    p <- length(ar.coefs)
    q <- length(ma.coefs)
    z <- rnorm(n+burn+p+q,sd=sqrt(innovar))
    x <- filter(z,c(1,ma.coefs),method="convolution",sides=1)
    x <- x[(q+1):(q+n+burn+p)]
    y <- x[1:p]
    for(t in (p+1):(p+n+burn))
    {
        next.y <- sum(ar.coefs*y[(t-1):(t-p)]) + x[t]
        y <- c(y,next.y)
    }
    y <- y[(p+burn+1):(p+burn+n)]
    return(y)
}

theta <- .7
sigma <- 1
n <- 100
x.sim <- armapq.sim(n,0,NULL,theta,sigma^2)
```

- We compute the Profile Whittle likelihood for a range of θ_1 values, and plot.
- We plot in log scale, for easier visualization.
- This function is lowest at the estimate $\widehat{\theta}_1$. This can be different from the true value $\theta_1 = .7$, due to statistical error.

```
pgram <- pgram.dft(x.sim)
thetas <- seq(-1000,1000)/1000
liks <- NULL
for(theta in thetas)
{
    liks <- c(liks,whittleprof.ma(theta,pgram))
}
plot(ts(log(liks[-c(1,2001)]),start=-1,frequency=1000))
```




```
thetas[which.min(liks[-c(1,2001)])+1]
```

```
## [1] 0.703
```

Exercise 10.46. Fitting via the Whittle Likelihood

- We now use the Profile Whitte likelihood to fit an MA(1) model to the Non-Defense Capitalization data.
- We wrap the objective function with an optimizer, using the BFGS method.

```
pgram <- pgram.dft(diff(ndc))
theta.init <- 0
fit.ma1 <- optim(theta.init,whittleprof.ma,pgram=pgram,method="BFGS")
theta.ope <- fit.ma1$par
sig2.ope <- fit.ma1$value
print(c(theta.ope,sig2.ope))
```

```
## [1] -0.498164132  0.005165085
```

Lesson 10-6: Computation of Optimal Predictors

- Calculating the predictors and the Gaussian likelihood is important.
- We explore algorithms for efficient computation.

Remark 10.6.1. Computational Efficiency in R

- Speed: for time series longer than $n = 1000$, direct inversion of covariance matrices becomes expensive, and it is important to use efficient algorithms.
- Memory: large data sets take up space (memory), so we only want to retain those portions that are needed.
- Recursive algorithms use less arithmetical operations (faster) and store fewer elements (less memory needed).

Definition 10.6.4.

- The k-fold *predictors* are minimizers of the linear prediction MSE of X_{k+1} from $\underline{X}_k = [X_1, \ldots, X_k]'$. They are given by
$$\underline{\varphi}_k = \Pi_k \, \Gamma_k^{-1} \, \underline{\gamma}_k,$$
where Π_k is a permutation matrix that reverses a vector (top to bottom).
- Recall $\underline{\gamma}_k = [\gamma(1), \ldots, \gamma(k)]'$.
- So $\widehat{X}_{k+1} = \underline{\varphi}_k' \underline{X}_k$.
- The last elements of the vector $\underline{\phi}_k$ weight the most recent data, and the first elements weight the old data.

Proposition 10.6.9

- We can recursively compute k-fold predictors in terms of the PACF $\kappa(k+1)$ via
$$\underline{\varphi}_{k+1} = \Pi_{k+1} \begin{bmatrix} \Pi_k \, \underline{\varphi}_k - \underline{\varphi}_k \, \kappa(k+1) \\ \kappa(k+1) \end{bmatrix}.$$

Exercise 10.39. Recursive Predictors

- We write R code for the recursive computation of k-fold predictors.
- First we load the ARMAauto function.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
```

```
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}
```

- Our function takes the ACF as input. We also incorporate the PACF calculation in the same loop.

```
pred.k <- function(gamma,max.lag)
{
    kappa.vec <- gamma[2]/gamma[1]
    varphi.vec <- kappa.vec
    if(max.lag > 1) {
    for(k in 2:max.lag)
    {
        schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
        kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    } }
    return(varphi.vec)
```

```
}
```

- We illustrate on a cyclic AR(2) process.

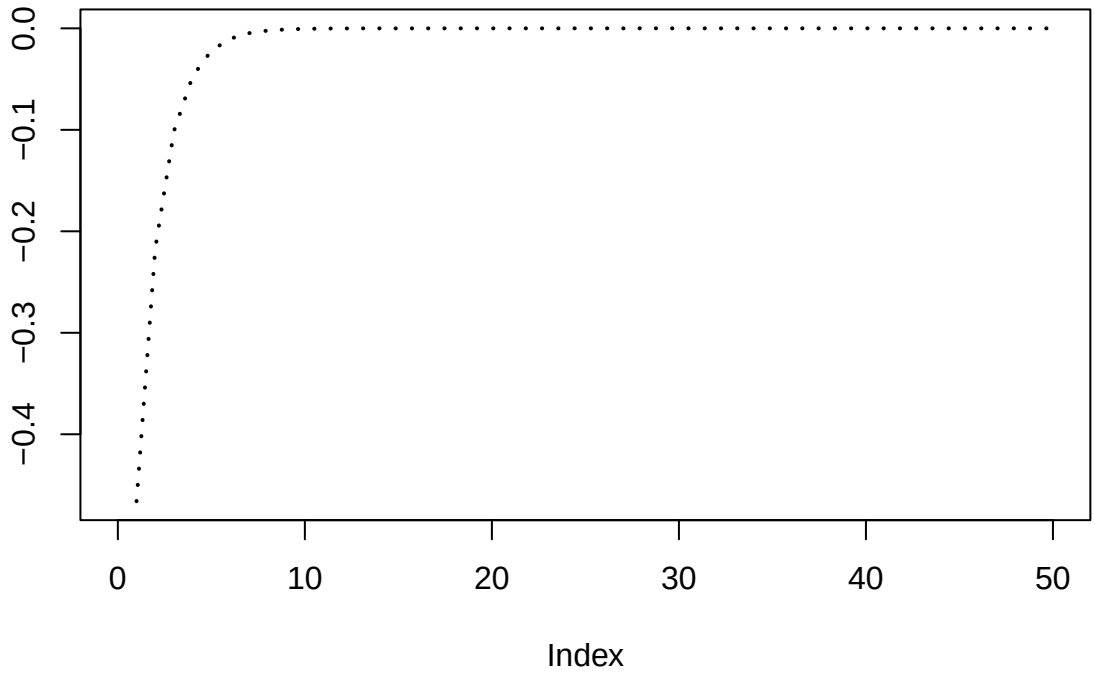
```
rho <- .8
omega <- pi/6
phi1 <- 2*rho*cos(omega)
phi2 <- -rho^2
gamma <- ARMAauto(c(phi1,phi2),NULL,20)
print(pred.k(gamma,20))
```

```
##  [1]  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
##  [8]  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## [15]  0.000000  0.000000  0.000000  0.000000 -0.640000  1.385641
```

Example 10.6.11. Predictors for Non-Defense Capitalization

- We compute the predictors for the MA(1) model fitted to the Non-Defense Capitalization data.
- As in Exercise 10.34, we have $\theta_1 = -.466$ and $\sigma^2 = .0053$.

```
theta <- -.466
sig.var <- .0053
max.lag <- 50
gamma <- rep(0,max.lag+1)
gamma[1] <- (1 +theta^2)*sig.var
gamma[2] <- theta*sig.var
varphi <- pred.k(gamma,max.lag)
plot(ts(c(NA,rev(varphi)),start=0),lty=3,lwd=2,xlab="Index",ylab="")
```



Lesson 10-7: The Gaussian Likelihood

- We examine the Gaussian likelihood further, and its efficient computation.

Paradigm 10.7.1. The Durbin-Levinson Algorithm

- We want an efficient way to compute the divergence, or the profile likelihood.
- The *Durbin-Levinson algorithm* gives efficient computation.
- How does it work? It can be shown that

$$\underline{X}_n' \, \Gamma_n^{-1} \, \underline{X}_n = \sum_{k=1}^{n} \epsilon_k^2 / d_k,$$

  where ϵ_k are in-sample forecast errors and d_k are their variances.
- These are computed recursively as follows:

$$\epsilon_{k+1} = X_{k+1} - \underline{\varphi}_k' \underline{X}_k$$
$$d_{k+1} = \gamma(0) - \underline{\varphi}_k' \Gamma_k \underline{\varphi}_k.$$

- In the first equation, this is $\epsilon_{k+1} = X_{k+1} - \widehat{X}_{k+1}$, the one-step-ahead forecast error.
- In the second equation, it can be shown this variance is always positive.
- We get both updates given $\underline{\varphi}_k$.

Exercise 10.41. Durbin-Levinson Algorithm

- We encode the Durbin-Levinson algorithm.
- The outputs are the likelihood, profile likelihood, the sum of squared residuals, and the residuals.

```
dl.alg <- function(gamma,data)
{
    # assumes that n > 1
    # assumes gamma includs lags 0 through n
    n <- length(data)
    kappa.vec <- gamma[2]/gamma[1]
    varphi.vec <- kappa.vec
    quad.new <- 0
    quads <- NULL
    ldet <- 0
    ldets <- NULL
    eps <- NULL

    for(k in 2:n)
    {
        schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
        eps.new <- data[k] - t(varphi.vec) %*% data[1:(k-1)]
        eps <- c(eps,eps.new)
        quad.new <- quad.new + eps.new^2/schur.new
        quads <- c(quads,quad.new)
        ldet <- ldet + log(schur.new)
        ldets <- c(ldets,ldet)
        kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    }
    gauss.lik <- n*log(2*pi) + quads[n-1] + ldets[n-1]
    gauss.likprof <- n*(1 + log(2*pi)) + n*log(quads[n-1]/n) + ldets[n-1]

    return(list(gauss.lik,gauss.likprof,quads[n-1]/n,eps))
}
```

- We test this on the Non-Defense Capitalization data, with a fitted MA(1) model.
- As in Exercise 10.34, we have $\theta_1 = -.466$ and $\sigma^2 = .0053$.

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
ndc.diff <- diff(ndc)
n <- length(ndc.diff)
theta <- -.466
sig.var <- .0053
gamma <- c(1+theta^2,theta,rep(0,n-1))*sig.var
print(dl.alg(gamma,ndc.diff)[[1]])
```

```
## [1] -702.4264
```

Exercise 10.47. Fitting an MA(1) Model to Non-Defense Capitalization

- We will use the profile likelihood based on the Durbin-Levinson algorithm to fit an MA(1) model to the Non-Defense Capitalization series.
- We have to load some R functions: polynomial multiplication and ARMA autocovariance calculation.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}
```

- Next we load functions that ensure stable parameterizing of ARMA polynomials.

```
psi2phi <- function(psi)
{
    p <- length(psi)
    pacfs <- (exp(psi)-1)/(exp(psi)+1)
    if(p==1)
    {
        phi <- pacfs
    } else
    {
        phi <- as.vector(pacfs[1])
        for(j in 2:p)
        {
            A.mat <- diag(j-1) - pacfs[j]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- A.mat %*% phi
            phi <- c(phi,pacfs[j])
        }
    }
    return(phi)
}

phi2psi <- function(phi)
{
    p <- length(phi)
    pacfs <- phi[p]
    if(p > 1)
    {
        phi <- as.vector(phi[-p])
        for(j in p:2)
        {
            A.mat <- diag(j-1) - pacfs[1]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- solve(A.mat,phi)
            pacfs <- c(phi[j-1],pacfs)
            phi <- phi[-(j-1)]
        }
    }
    psi <- log(1+pacfs) - log(1-pacfs)
    return(psi)
}
```

- Next we write the profile likelihood function.

```
likprof.ma <- function(psi,data)
{
    n <- length(data)
    theta <- -1*psi2phi(psi)
    gamma <- ARMAauto(NULL,theta,n)
    lik <- dl.alg(gamma,data)[[2]]
    return(lik)
}
```

- Now we are ready to do the fitting.

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
```

```
ndc.diff <- diff(ndc)
n <- length(ndc.diff)
psi.init <- 0
fit.ma1 <- optim(psi.init,likprof.ma,data=ndc.diff,method="BFGS")
psi.mle <- fit.ma1$par
theta.mle <- -1*psi2phi(psi.mle)
gamma <- ARMAauto(NULL,theta.mle,n)
sig2.mle <- dl.alg(gamma,ndc.diff)[[3]]
```

- The results are: $\widehat{\theta_1}$ = -0.4911631 and $\widehat{\sigma^2}$ = 0.0051809.

Lesson 10-8: Model Assessment

- Once we have fitted a model, we need to assess the fit.
- We should also compare to competing models.

Remark 10.8.1. Computing Residuals

- If an ARMA model was fitted using the Gaussian divergence, we get the one-step-ahead forecast errors (in-sample) as output of the Durbin-Levinson algorithm. These are *time series residuals*.
- If our model is a good fit, the time series residuals should resemble white noise.

Example 10.8.2. Residuals of MA(1) Model Fitted to Non-Defense Capitalization

- We continue Exercise 10.47 of previous lesson. First load all the needed functions.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
```

```
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}

dl.alg <- function(gamma,data)
{
    # assumes that n > 1
    # assumes gamma includs lags 0 through n
    n <- length(data)
    kappa.vec <- gamma[2]/gamma[1]
    varphi.vec <- kappa.vec
    quad.new <- 0
    quads <- NULL
    ldet <- 0
    ldets <- NULL
    eps <- NULL

    for(k in 2:n)
    {
        schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
        eps.new <- data[k] - t(varphi.vec) %*% data[1:(k-1)]
        eps <- c(eps,eps.new)
        quad.new <- quad.new + eps.new^2/schur.new
        quads <- c(quads,quad.new)
        ldet <- ldet + log(schur.new)
        ldets <- c(ldets,ldet)
        kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
```

```
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    }
    gauss.lik <- n*log(2*pi) + quads[n-1] + ldets[n-1]
    gauss.likprof <- n*(1 + log(2*pi)) + n*log(quads[n-1]/n) + ldets[n-1]

    return(list(gauss.lik,gauss.likprof,quads[n-1]/n,eps))
}

psi2phi <- function(psi)
{
    p <- length(psi)
    pacfs <- (exp(psi)-1)/(exp(psi)+1)
    if(p==1)
    {
        phi <- pacfs
    } else
    {
        phi <- as.vector(pacfs[1])
        for(j in 2:p)
        {
            A.mat <- diag(j-1) - pacfs[j]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- A.mat %*% phi
            phi <- c(phi,pacfs[j])
        }
    }
    return(phi)
}

phi2psi <- function(phi)
{
    p <- length(phi)
    pacfs <- phi[p]
    if(p > 1)
    {
        phi <- as.vector(phi[-p])
        for(j in p:2)
        {
            A.mat <- diag(j-1) - pacfs[1]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- solve(A.mat,phi)
            pacfs <- c(phi[j-1],pacfs)
            phi <- phi[-(j-1)]
        }
    }
    psi <- log(1+pacfs) - log(1-pacfs)
    return(psi)
}
```

- We use a profile likelihood for an ARMA process.

```
likprof.arma <- function(psi,q,data)
{
    n <- length(data)
    r <- length(psi)
    p <- r-q
```

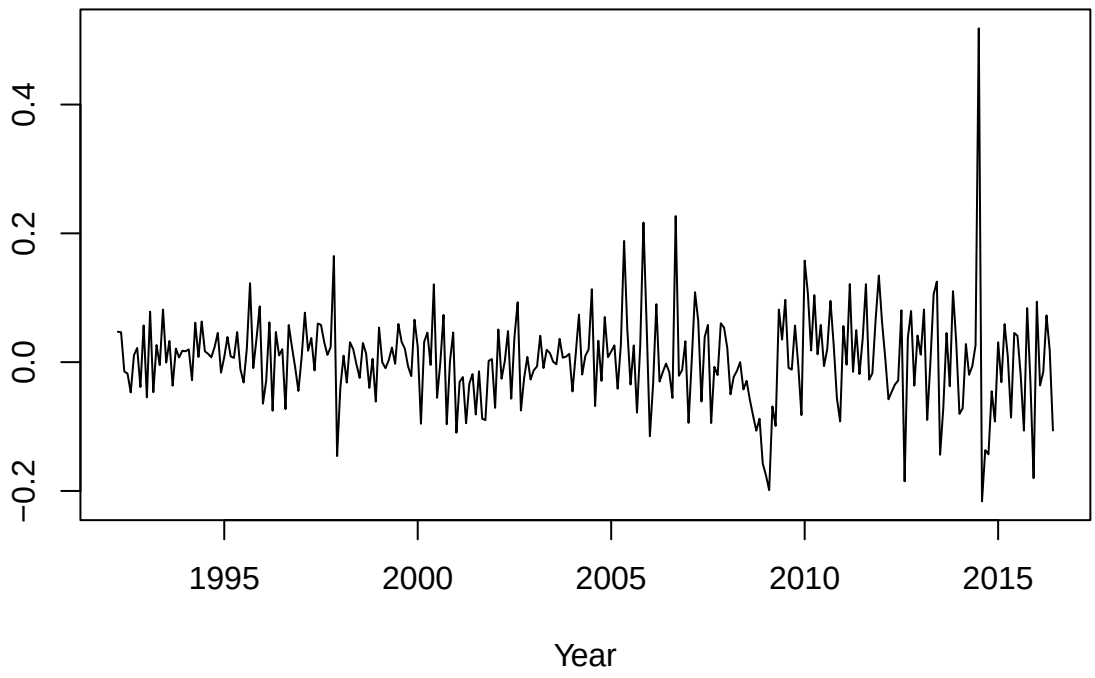
```
    phi <- NULL
    theta <- NULL
    if(p > 0) { phi <- psi2phi(psi[1:p]) }
    if(q > 0) { theta <- -1*psi2phi(psi[(p+1):(p+q)]) }
    gamma <- ARMAauto(phi,theta,n)
    lik <- dl.alg(gamma,data)[[2]]
    return(lik)
}
```

- We can fit the series as before.

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
ndc.diff <- diff(ndc)
n <- length(ndc.diff)
psi.init <- 0
fit.ma1 <- optim(psi.init,likprof.arma,q=1,data=ndc.diff,method="BFGS")
psi.mle <- fit.ma1$par
theta.mle <- -1*psi2phi(psi.mle)
gamma <- ARMAauto(NULL,theta.mle,n)
sig2.mle <- dl.alg(gamma,ndc.diff)[[3]]
```

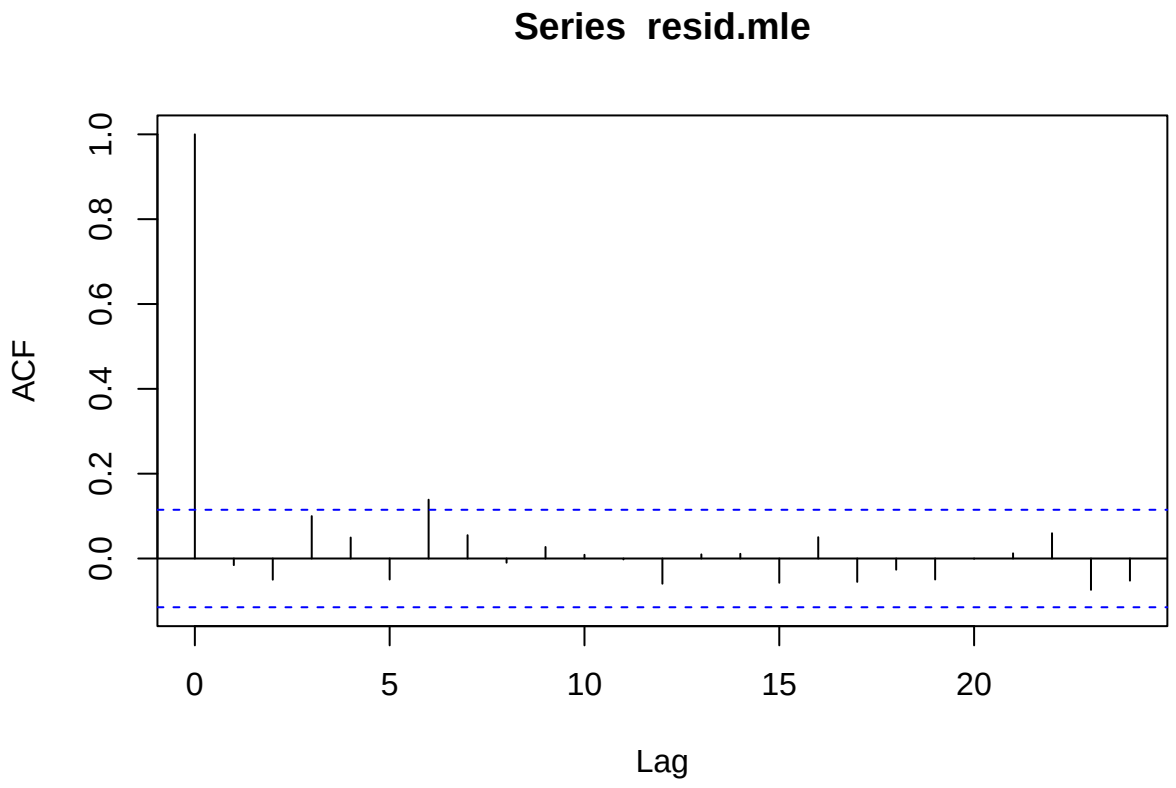
- So the parameter estimates are $\widehat{\theta_1}$ = -0.4911631 and $\widehat{\sigma^2}$ = 0.0051809.
- We can get the residuals as output as well.

```
resid.mle <- dl.alg(gamma,ndc.diff)[[4]]
plot(ts(resid.mle,start=c(1992,4),frequency=12),
     xlab="Year",ylab="",lwd=1)
```



- Some structure in the residuals is evident around 2008 (the nadir of the Great Recession).
- We can examine the serial correlation structure using the sample acf (and pacf):

```
acf(resid.mle)
```



Remark 10.8.6. Portmanteau Statistics

- To assess whether the residuals resemble white noise, we can measure the autocovariances.
- We can combine into a single measure by taking a sum of squared autocorrelations.
- This is an aggregate measure, like a *portmanteau* (suitcase) of statistics.

Definition 10.8.7.

- The *total variation* of a stationary time series with absolutely summable acvf is

$$\zeta = \sum_{k \neq 0} \gamma(k)^2.$$

- Note that $\zeta = 0$ if and only if the process is white noise.

Remark 10.8.9. Testing Total Variation

- So we can test whether a process is white noise by testing $\zeta = 0$, using the test statistic

$$\widehat{\zeta} = \frac{1}{2n} \sum_{\ell=-[n/2]-n+1}^{[n/2]} I(\lambda_\ell)^2 - \widehat{\gamma}(0)^2.$$

- It can be shown that

$$\sqrt{n}\,\widehat{\zeta} \Rightarrow \mathcal{N}(0, 2\sigma^8)$$

under $H_0 : \zeta = 0$, where σ^2 is the variance of the process.

- A normalized test statistic that is asymptotically standard normal is given by

$$\sqrt{n/2}\,\frac{\widehat{\zeta}}{\widehat{\gamma}(0)^2}.$$

- We perform an upper one-sided test; large values indicate that serial correlation is present.
- We can apply this to the time series residuals.

Example 10.8.10. Total Variation White Noise Testing of Non-Defense Capitalization

- We apply the total variation test statistic to the time series residuals from the fit of an MA(1) to Non-Defense Capitalization.

```
n <- length(resid.mle)
gamma.hat <- acf(resid.mle,lag.max=n,plot=FALSE,type="covariance")$acf
lambda <- seq(-n/2+1,n/2)*2*pi/n
pgram <- cos(0*lambda)*gamma.hat[1]
for(h in 1:(n-1))
{
    pgram <- pgram + 2*cos(h*lambda)*gamma.hat[h+1]
}
pgram <- ts(pgram,start=0,frequency=n)
tot.var <- .5*mean(pgram^2) - gamma.hat[1]^2
tstat <- sqrt(n)*tot.var/sqrt(2*gamma.hat[1]^4)
pval <- 1-pnorm(tstat)
```

- The test statistic is $-3.1966361 \times 10^{-6}$, and the studentized statistic is -1.4351016, with asymptotic p-value 0.9243709.
- So there is no evidence for rejecting H_0, and we conclude that no serial correlation remains.

Lesson 10-9: Information Criteria

- We discuss how to sift competing models.

Remark 10.9.1. Underfitting and Overfitting

- *Underfitting* is specifying a wrong model, such that the model order is too low. Then the parameter estimates are not consistent, and there is bias.
- *Overfitting* is specifying a model with too high of a model order (or unneeded parameters). However, the true model is still nested within the specified model. In this case the parameter estimates are inefficient (their variance is higher than it would be if the model were correctly specified).
- Whenever we add model complexity, the Gaussian divergence decreases, and hence overfitting may arise.

Definition 10.9.2.

- Suppose $\mathcal{F}$ is a linear time series model with parameter $\underline{\omega}$.
- If restricting some of the elements of $\underline{\omega}$ to zero yields another model $\mathcal{G}$, we say that $\mathcal{G}$ is *nested* in $\mathcal{F}$, and $\mathcal{F}$ is *nests* in $\mathcal{G}$.

Example 10.9.3. AR, MA, and ARMA Nesting

- Any AR($p+r$) nests an AR(p), for $r > 0$: set the last r components of $\underline{\omega}$ to zero, and you get the parameter vector of an AR(p).

- Similarly for MA models.
- Similarly, an EXP($m+\ell$) nests an EXP(m) model.
- An ARMA(p,q) nests both an AR(p) and an MA(q) model.
- White noise is nested in all AR, MA, ARMA, and EXP models.

Remark 10.9.5. Occam's Razor and the Principle of Parsimony

- If two models fit the data equally well, the simpler one is preferred.
- We can measure simplicity through the number of parameters.
- We can penalize the selection of models with a high number of parameters.
- This helps to guard against overfitting.

Definition 10.9.6.

- The *Akaike Information Criterion* (AIC) for a linear model with r parameters (not including the input variance σ^2) is given by the Gaussian divergence plus $2r$:

$$\text{AIC}(\underline{\omega}, \sigma^2) = \mathcal{L}(\underline{\omega}, \sigma^2) + 2r.$$

- Hence, minimizing AIC protects against overfitting.
- Using the profile likelihood, minimizing AIC is equivalent to minimizing

$$n \log \widehat{\sigma}^2 + 2r.$$

- Note that $\widehat{\sigma}^2$ depends on r.

Remark 10.9.7. Information Criteria

- Other penalties can be used. For example, the *Bayesian Information Criterion* (BIC) has penalty of $2r \log n$ instead of $2r$.
- AIC has a tendency towards a slight over-parameterization. This can be a good thing, as a safeguard against model mis-specification.

Exercise 10.59. Information Criteria Model Comparison

- We fit an AR(2) model and an MA(1) model to the Non-Defense Capitalization series.
- We must load up some functions.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
```

```
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}

dl.alg <- function(gamma,data)
{
    # assumes that n > 1
    # assumes gamma includs lags 0 through n
    n <- length(data)
    kappa.vec <- gamma[2]/gamma[1]
    varphi.vec <- kappa.vec
    quad.new <- 0
    quads <- NULL
    ldet <- 0
    ldets <- NULL
    eps <- NULL

    for(k in 2:n)
    {
        schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
        eps.new <- data[k] - t(varphi.vec) %*% data[1:(k-1)]
```

```
        eps <- c(eps,eps.new)
        quad.new <- quad.new + eps.new^2/schur.new
        quads <- c(quads,quad.new)
        ldet <- ldet + log(schur.new)
        ldets <- c(ldets,ldet)
        kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    }
    gauss.lik <- n*log(2*pi) + quads[n-1] + ldets[n-1]
    gauss.likprof <- n*(1 + log(2*pi)) + n*log(quads[n-1]/n) + ldets[n-1]

    return(list(gauss.lik,gauss.likprof,quads[n-1]/n,eps))
}

psi2phi <- function(psi)
{
    p <- length(psi)
    pacfs <- (exp(psi)-1)/(exp(psi)+1)
    if(p==1)
    {
        phi <- pacfs
    } else
    {
        phi <- as.vector(pacfs[1])
        for(j in 2:p)
        {
            A.mat <- diag(j-1) - pacfs[j]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- A.mat %*% phi
            phi <- c(phi,pacfs[j])
        }
    }
    return(phi)
}

phi2psi <- function(phi)
{
    p <- length(phi)
    pacfs <- phi[p]
    if(p > 1)
    {
        phi <- as.vector(phi[-p])
        for(j in p:2)
        {
            A.mat <- diag(j-1) - pacfs[1]*diag(j-1)[,(j-1):1,drop=FALSE]
            phi <- solve(A.mat,phi)
            pacfs <- c(phi[j-1],pacfs)
            phi <- phi[-(j-1)]
        }
    }
```

```
    psi <- log(1+pacfs) - log(1-pacfs)
    return(psi)
}

likprof.arma <- function(psi,q,data)
{
    n <- length(data)
    r <- length(psi)
    p <- r-q
    phi <- NULL
    theta <- NULL
    if(p > 0) { phi <- psi2phi(psi[1:p]) }
    if(q > 0) { theta <- -1*psi2phi(psi[(p+1):(p+q)]) }
    gamma <- ARMAauto(phi,theta,n)
    lik <- dl.alg(gamma,data)[[2]]
    return(lik)
}
```

- We use the Gaussian likelihood to fit each model, and compare the fits via AIC and BIC.

```
ndc <- read.table("Nondefcap.dat")
ndc <- ts(ndc[,2],start=c(1992,3),frequency=12,names= "NewOrders")
ndc.diff <- diff(ndc)
n <- length(ndc.diff)
gamma.hat <- acf(ndc.diff,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
phi.init <- solve(toeplitz(gamma.hat[1:2]),gamma.hat[2:3])
```

- First we get Yule-Walker estimates (-0.4841623, -0.2650735) to initialize the MLE optimization.
- Then we find the MLEs.

```
psi.init <- phi2psi(phi.init)
fit.ar2 <- optim(psi.init,likprof.arma,q=0,data=ndc.diff,method="BFGS")
```

- Second, we get the MLEs for the MA(1) model, intializing at zero.

```
psi.init <- 0
fit.ma1 <- optim(psi.init,likprof.arma,q=1,data=ndc.diff,method="BFGS")
```

- We compute the AIC and BIC values next.

```
lik.ar2 <- fit.ar2$value
lik.ma1 <- fit.ma1$value
aic.ar2 <- lik.ar2 + 2*2
bic.ar2 <- lik.ar2 + 2*2*log(n)
aic.ma1 <- lik.ma1 + 2*1
bic.ma1 <- lik.ma1 + 2*1*log(n)
```

- The AIC for the MA(1) and AR(2) models is -706.0108666 and -707.1981055 respectively.
- The BIC for the MA(1) and AR(2) models is -696.657359 and -688.4910902 respectively.
- So according to AIC the AR(2) model is preferred, but by BIC the MA(1) model is preferred!

Lesson 10-10: Model Comparisons

- We can also compare two models using test statistics.
- Commonly, we consider the likelihood ratio, or equivalently the difference of divergences.

Corollary 10.10.5.

- Suppose $\{X_t\}$ is a linear time series with i.i.d. inputs.
- Suppose two models are fitted, which are nested.
- The nested model parameter vector is denote $\underline{\omega}^0$ (with dimension s).
- The nesting model parameter vector is denote $\underline{\omega}$ (with dimension r).
- We assume some other technical conditions.
- If the nested model is correct (it is the model of the process), then the likelihood ratio statistic

$$\mathcal{L}(\widehat{\underline{\omega}^0}, \widehat{\sigma}^2) - \mathcal{L}(\widehat{\underline{\omega}}, \widehat{\sigma}^2)$$

  is asymptotically χ^2_{r-s}.
- We can use the profile likelihood, and compute the above statistic by taking the difference of logged innovation variance estimates, multiplied by sample size.

Exercise 10.58. Nested Model Comparisons

- We fit an AR(1) and AR(2) model to the growth rate of the U.S. population data. Which model is better?
- The MLEs have the same asymptotic behavior as the Yule-Walker estimates, when fitting AR models.
- So we load up code for Yule-Walker estimation.

```
arp.fityw <- function(data,p)
{
    gamma.hat <- acf(data,lag=p,type="covariance",plot=FALSE)$acf[,,1]
    phi <- solve(toeplitz(gamma.hat[1:p]),gamma.hat[2:(p+1)])
    sigma2 <- gamma.hat[1] - sum(phi*gamma.hat[2:(p+1)])
    hess <- sigma2*diag(solve(toeplitz(gamma.hat[1:p])))
    return(list(phi,sigma2,hess))
}
```

- Then we fit both models to the growth rate.

```
pop <- read.table("USpop.dat")
pop <- ts(pop, start = 1901)
pop.gr <- diff(pop)
n <- length(pop.gr)
```

- We compute the likelihood ratio test statistic by n times the difference of $\log \widehat{\sigma}^2$ for each model.

```
sig2.ar1 <- arp.fityw(pop.gr,1)[[2]]
sig2.ar2 <- arp.fityw(pop.gr,2)[[2]]
glr <- n*(log(sig2.ar1) - log(sig2.ar2))
print(c(glr,1-pchisq(glr,df=1)))
```

```
## [1] 0.4016207 0.5262534
```

- The large p-value indicates that we fail to reject, and hence the AR(1) model is best.

Lesson 10-11: Iterative Forecasting

- With a fitted model we can compute forecasts, backcasts, and missing values using the normal equations.
- The fitted model provides us with estimated autocovariances, which are needed in the normal equations.

Proposition 10.11.3

- The h-step-ahead predictor can be calculated in terms of j-step-ahead predictors, for $1 \leq j < h$.

- Get the 2-step-ahead from the 1-step-ahead predictor as follows:

$$\widehat{X}_{n+1} = \underline{\varphi}_n' \underline{X}_n$$
$$\widehat{X}_{n+2} = \underline{\varphi}_{n+1}' \begin{bmatrix} \underline{X}_n \\ \widehat{X}_{n+1} \end{bmatrix}.$$

- Continue on iteratively to get the h-step-ahead: append the forecast and apply the next one-step-ahead predictor.

Remark 10.11.4. Iterative Forecasting is Efficient

- Direct calculation from the normal equations of forecasts would be expensive, because we have to invert a n-dimensional covariance matrix.
- The one-step-ahead predictors $\underline{\varphi}_n$ are efficiently computed in the Durbin-Levinson algorithm, so the iterative approach of Proposition 10.11.3 is faster.

Paradigm 10.11.7. Forecasting an Integrated Process

- Suppose that $\{X_t\}$ is a nonstationary time series that is integrated, and we want to forecast it.
- Suppose that $\delta(B) = 1 - \sum_{j=1}^{d} \delta_j B^j$ is a *differencing* polynomial that reduces $\{X_t\}$ to stationarity, i.e.,

$$Y_t = \delta(B) X_t = X_t - \sum_{j=1}^{d} \delta_j X_{t-j}$$

  is a stationary process.

- Then we can forecast $\{X_t\}$ as follows: forecast the $\{Y_t\}$ series, obtaining $\widehat{Y}_{n+1}$, and use the formula

$$\widehat{X}_{n+1} = \sum_{j=1}^{d} \delta_j X_{n+1-j} + \widehat{Y}_{n+1}.$$

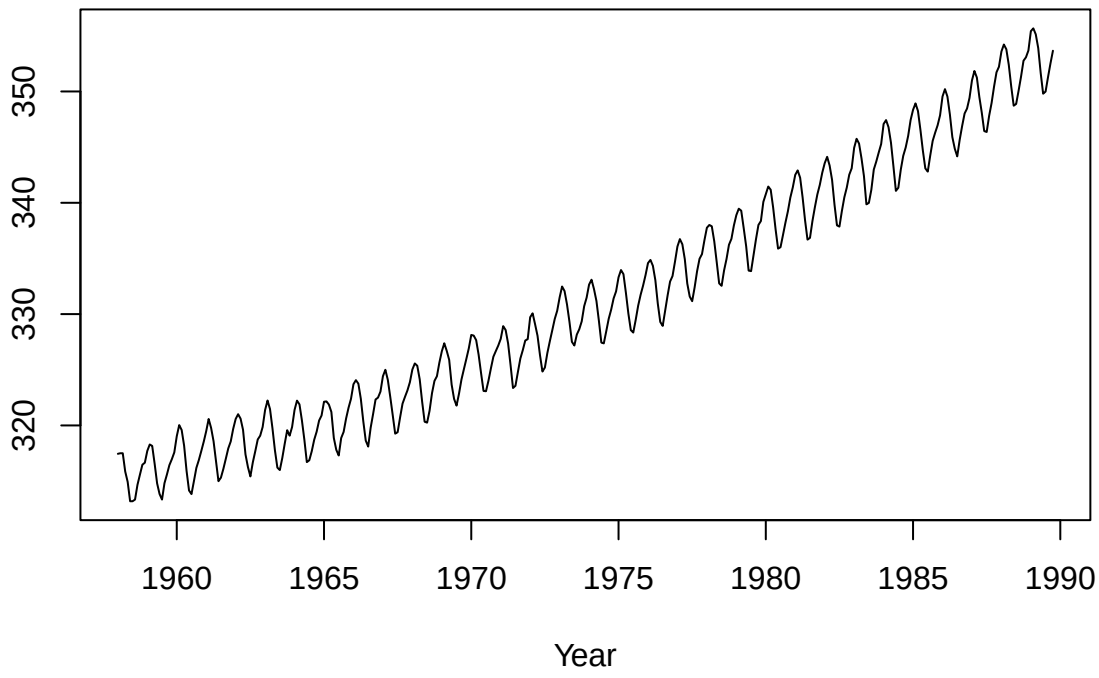
- This works when $X_n, \ldots, X_{n+1-d}$ are observed.

Exercise 10.61. Recursive Forecasting of Mauna Loa

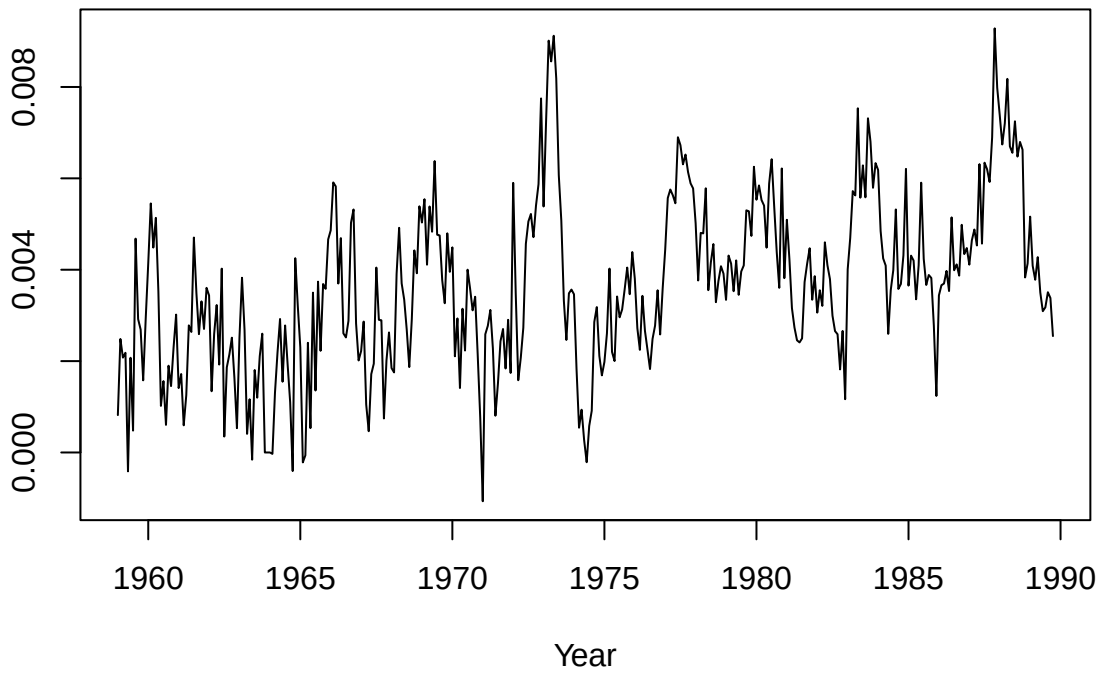
- We will forecast the Mauna Loa time series.
- We first fit an AR(p) model to the annual differences of the logged data.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
```

```
    if (p > 0)
    {
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}

mau <- read.table("mauna.dat",header=TRUE,sep="")
mau <- ts(mau,start=1958,frequency=12)
mau.gr <- diff(log(mau),lag=12)
plot(mau,ylab="",xlab="Year")
```

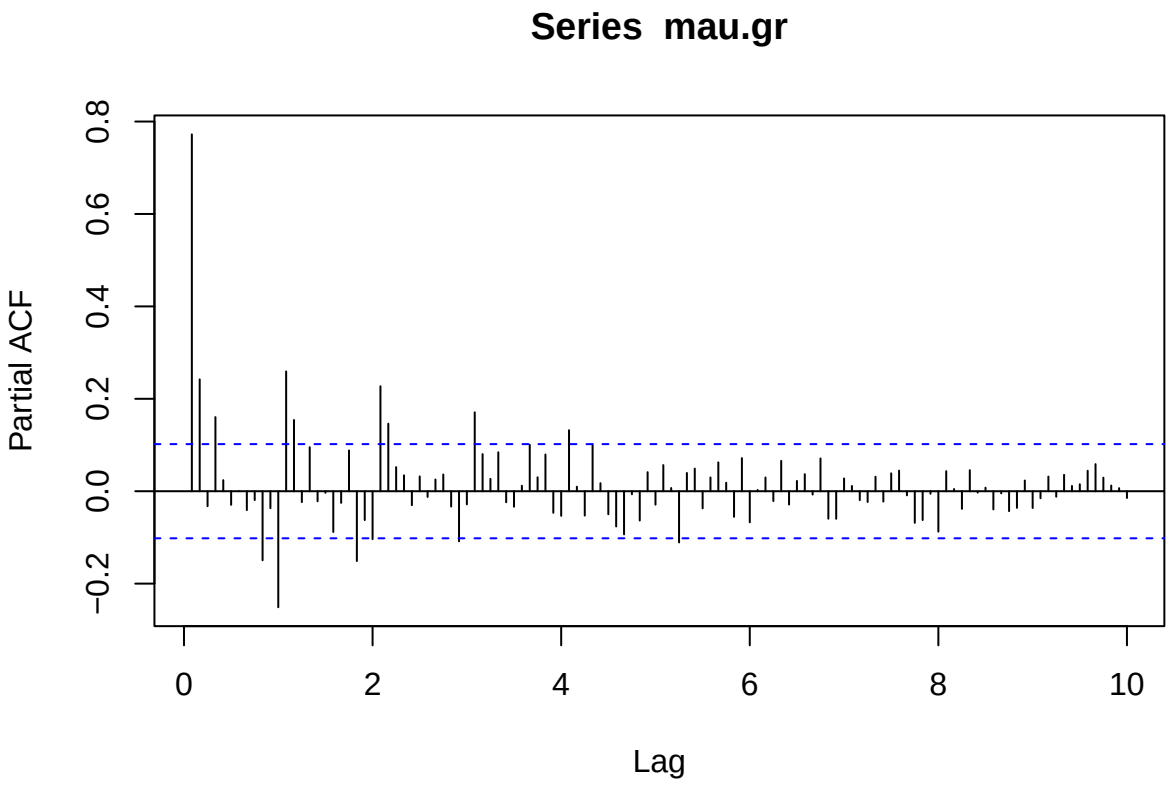



```
plot(mau.gr,ylab="",xlab="Year")
```



- We also examine the pacf.

```
delta <- c(1,rep(0,11),-1)
d <- length(delta)-1
n <- length(mau.gr)
mau.pacf <- pacf(mau.gr,lag.max=120)
```



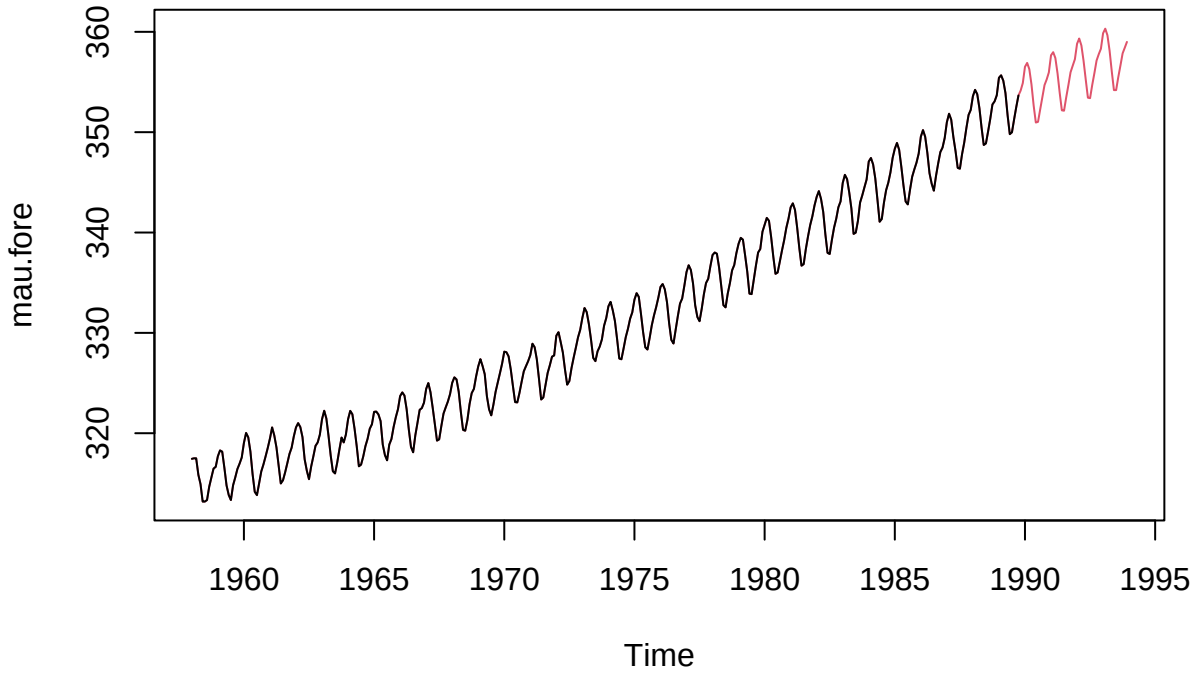
- By inspection of the pacf plot, we choose $p = 49$. We fit an AR(49) model using Yule-Walker.

```
arp.fityw <- function(data,p)
{
    gamma.hat <- acf(data,lag=p,type="covariance",plot=FALSE)$acf[,,1]
    phi <- solve(toeplitz(gamma.hat[1:p]),gamma.hat[2:(p+1)])
    sigma2 <- gamma.hat[1] - sum(phi*gamma.hat[2:(p+1)])
    hess <- sigma2*diag(solve(toeplitz(gamma.hat[1:p])))
    return(list(phi,sigma2,hess))
}

p.hat <- 49
ar.coef <- arp.fityw(mau.gr,p.hat)[[1]]
sigma2 <- arp.fityw(mau.gr,p.hat)[[2]]
```

- Then we forecast 50 steps ahead. The basic pattern gets extended.

```
pred.k <- function(gamma,max.lag)
{
    kappa.vec <- gamma[2]/gamma[1]
    varphi.vec <- kappa.vec
    if(max.lag > 1) {
    for(k in 2:max.lag)
    {
        schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
        kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
```

```
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    } }
    return(varphi.vec)
}

H <- 50
gamma <- ARMAauto(ar.coef,NULL,n+H)*sigma2
x <- mau.gr
for(k in 1:H)
{
    varphi.vec <- pred.k(gamma,n+k-1)
    cast <- sum(varphi.vec * x)
    x <- c(x,cast)
}
mau.fore <- c(log(mau)[1:d],filter(x,-delta[-1],method="recursive",init=log(mau)[d:1]))
mau.fore <- ts(exp(mau.fore),start=1958,frequency=12)

plot(mau.fore,col=2)
lines(mau)
```



Lesson 10-12: Imputation and Signal Extraction

- By forecasting and backcasting, we can extend a time series and then apply filters.

Remark 10.12.4. Extending a Time Series Sample

- Recall from Remark 3.1.7 that applying a moving average filter leaves out the beginning and end of the sample.
- We can extend a time series via forecasts and backcasts, and then apply the filter to fix this problem.
- If the symmetric filter is of length $2m+1$, we need m forecasts and m backcasts.

Proposition 10.12.6

- Suppose a filter $\Psi(B)$ is applied to $\{X_t\}$, yielding $\{Y_t\}$.
- Given the sample $\underline{X}_n$, for any time t

$$\mathbb{E}[Y_t|\underline{X}_n] = \sum_{j=-\infty}^{\infty} \psi_j \, \mathbb{E}[X_{t-j}|\underline{X}_n].$$

- This justifies extending a series: we filter the sample $\underline{X}_n$ extended to points $t > n$ and $t < 1$ via appending forecast and backcasts.

Example 10.12.8. HP Trend of West Starts Annual Rate

- We consider the annual growth rate of the West Housing Starts data.
- We apply the Hodrick-Prescott (HP) filter.
- First, we will fit an MA(12) model.

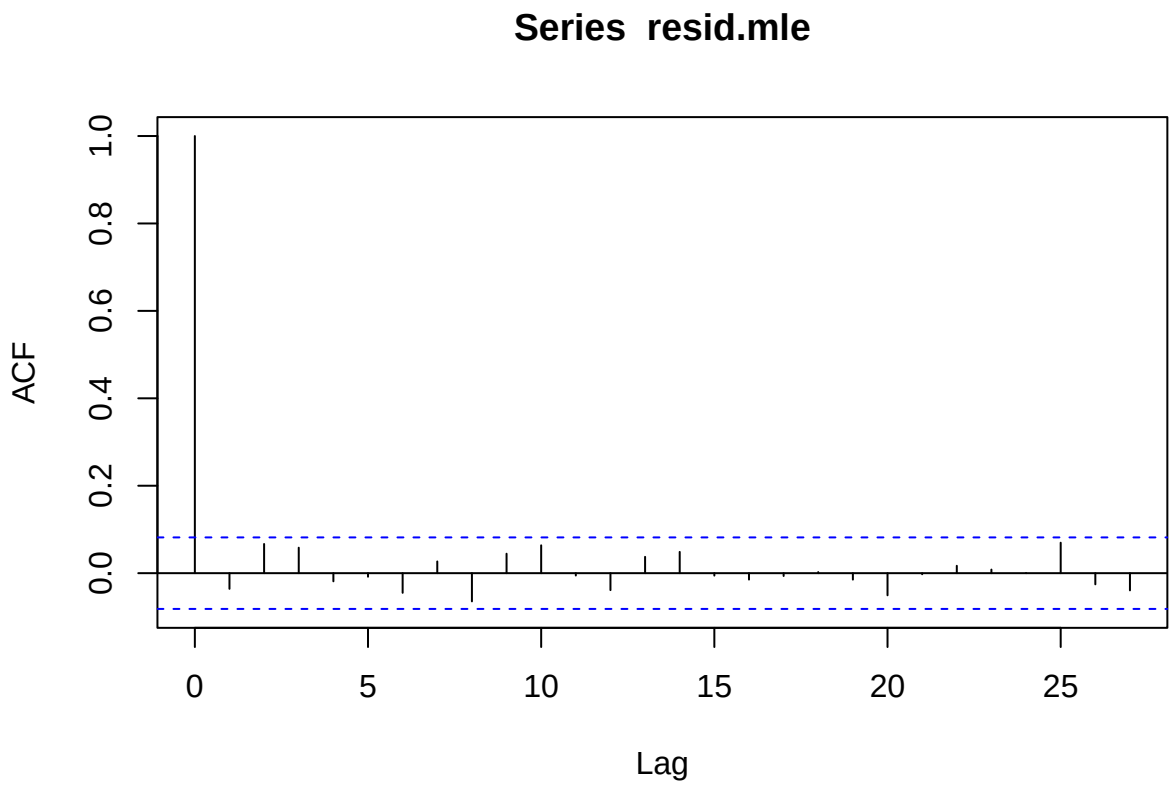
```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
```

```
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}

Wstarts <- read.table("Wstarts.b1",header=TRUE,skip=2)[,2]
Wstarts <- ts(Wstarts,start = 1964,frequency=12)
west <- diff(log(Wstarts),lag=12)
n <- length(west)
gamma.hat <- acf(west,lag=n-1,type="covariance",plot=FALSE)$acf[,,1]
q <- 12

gausslik.maq <- function(theta,data,fit.flag)
{
    n <- length(data)
    gamma.maq <- ARMAauto(NULL,theta,n)
    kappa.vec <- gamma.maq[2]/gamma.maq[1]
    varphi.vec <- kappa.vec
    quad.new <- 0
    quads <- NULL
    ldet <- 0
    ldets <- NULL
    eps <- NULL

    for(k in 2:n)
    {
        schur.new <- gamma.maq[1] - t(varphi.vec) %*% rev(gamma.maq[2:k])
        eps.new <- data[k] - t(varphi.vec) %*% data[1:(k-1)]
        eps <- c(eps,eps.new)
        quad.new <- quad.new + eps.new^2/schur.new
        quads <- c(quads,quad.new)
```

```
        ldet <- ldet + log(schur.new)
        ldets <- c(ldets,ldet)
        kappa.new <- (gamma.maq[k+1] - sum(varphi.vec*gamma.maq[2:k]))/schur.new
        kappa.new <- kappa.new[1,1]
        if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
        kappa.vec <- c(kappa.vec,kappa.new)
        varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
        varphi.vec <- rev(c(varphi.vec,kappa.new))
    }
    gauss.lik <- n*(1 + log(2*pi)) + n*log(quads[n-1]/n) + ldets[n-1]

    if(fit.flag) { return(gauss.lik) } else
    {
        return(list(quads[n-1]/n,eps))
    }
}
```

- Because the fitting takes some time, we instead input the MLE directly below (based on a prior run).

```
theta.mle <- c(0.651972958905815, 0.652200820985369, 0.627422633499361, 0.646525805480097,
0.625424457604761, 0.644635978951906, 0.637430978594635, 0.635720014729216,
0.625268961947794, 0.634947219500035, 0.600750521900405, -0.367624464655481
)
sig2.mle <- gausslik.maq(theta.mle,west,fit.flag=FALSE)[[1]]
resid.mle <- gausslik.maq(theta.mle,west,fit.flag=FALSE)[[2]]
acf(resid.mle)
```



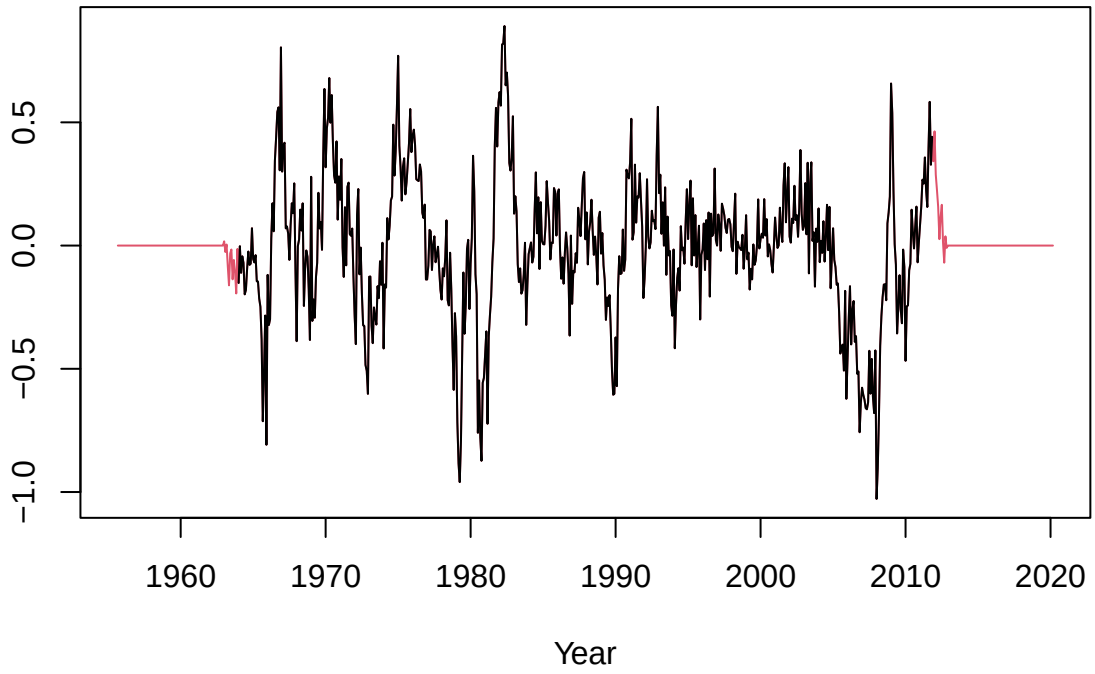
- The ACF of the residuals indicates this model fit seems adequate.
- Next, we obtain an invertible form of the moving average polynomial by root flipping.

```
theta.poly <- c(1,theta.mle)
new.poly <- 1
sig2.new <- sig2.mle
ma.roots <- polyroot(theta.poly)
for(i in 1:q)
{
    if(Mod(ma.roots[i]) < 1)
    {
        ma.roots[i] <- 1/ma.roots[i]
        sig2.new <- sig2.new*(1/ma.roots[i]^2)
    }
    new.poly <- polymul(new.poly,c(1,-1/ma.roots[i]))
}
new.poly <- Re(new.poly)
sig2.new <- Re(sig2.new)
theta.inv <- new.poly[-1]
```

- Then we extend the time series with forecasts and backcasts.
- We go 100 steps forwards and backwards.

```
H <- 100
T <- n+2*H
gamma <- ARMAauto(NULL,theta.inv,T)*sig2.new
x <- c(rep(NA,H),west,rep(NA,H))
x.fore <- c(west,rep(NA,H))
T <- n+H
kappa.vec <- gamma[2]/gamma[1]
varphi.vec <- kappa.vec
for(k in 2:(T-1))
{
    schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
    kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
    kappa.new <- kappa.new[1,1]
    if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
    kappa.vec <- c(kappa.vec,kappa.new)
    varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
    varphi.vec <- rev(c(varphi.vec,kappa.new))
    cast <- sum(varphi.vec * x.fore[1:k])
    if(k >= n) x.fore[k+1] <- cast
}
x.aft <- rev(c(rep(NA,H),x.fore))
T <- n+2*H
kappa.vec <- gamma[2]/gamma[1]
varphi.vec <- kappa.vec
for(k in 2:(T-1))
{
    schur.new <- gamma[1] - t(varphi.vec) %*% rev(gamma[2:k])
    kappa.new <- (gamma[k+1] - sum(varphi.vec*gamma[2:k]))/schur.new
    kappa.new <- kappa.new[1,1]
    if(abs(kappa.new) < 10^(-15)) kappa.new <- 0
    kappa.vec <- c(kappa.vec,kappa.new)
    varphi.vec <- rev(varphi.vec) - kappa.new * varphi.vec
```

```
    varphi.vec <- rev(c(varphi.vec,kappa.new))
    cast <- sum(varphi.vec * x.aft[1:k])
    if(k >= n+H) x.aft[k+1] <- cast
}
x.cast <- rev(x.aft)
x.ext <- ts(x.cast,start=(1964-H/12),frequency=12)
plot(x.ext,col=2,ylab="",xlab="Year")
lines(ts(x,start=(1964-H/12),frequency=12))
```



- Then we filter the extended data.
- We use the HP filter with parameter 1/1600. This uses an exact formula for the filter (Exercise 6.29 in the book).

```
q <- 1/1600
s <- (2*q + 2*q^(1/2)*(q+16)^(1/2))^(1/2)
r <- (q^(1/2) + (q+16)^(1/2) + s)/4
c <- q/r^2
phi1 <- 2*(q^(1/2)-(q+16)^(1/2))/(4*r)
phi2 <- (q^(1/2)+(q+16)^(1/2) - s)/(4*r)
theta <- atan(s/4)
lags <- seq(0,H)
psi <- 2*c*r^(4-lags)*sin(theta)*(r^2*sin(theta*(1+lags)) - sin(theta*(lags-1)))
psi <- psi/((1-2*r^2*cos(2*theta)+r^4)*(r^2-1)*(1-cos(2*theta)))
psi <- c(rev(psi),psi[-1])
west.trend <- filter(x.ext,psi,method="convolution")[(H+1):(length(x.ext)-H)]
west.trend <- ts(west.trend,start=1964,frequency=12)
```

```
plot(x.ext,lwd=2,col=grey(.9),xlab="Year",ylab="Starts Annual Growth")
lines(ts(c(rep(NA,H),west,rep(NA,H)),start=1964-H/12,frequency=12),lwd=2,col=grey(.8))
lines(west.trend,col=1,lwd=2,lty=2)
```