

STAT 720 sp 2019 Lec 04

Karl Gregory

1/24/2019

Forecasting for time series

Given a length- n realization X_1, \dots, X_n of a stationary time series $\{X_t, t \in \mathbb{Z}\}$, we consider predicting the value of X_{n+h} , for $h \geq 1$, based on the values X_1, \dots, X_n . Specifically, we will consider a predictor $P_n X_{n+h}$ of X_{n+h} which is linear in the values X_1, \dots, X_n , taking the form

$$P_n X_{n+h} = a_0 + \sum_{i=1}^n a_i X_{n+1-i}.$$

Minimizing the expected squared prediction error

A desirable property of a predictor is that the expected or mean squared prediction error (MSPE) be small. The MSPE for the predictor we consider is given by

$$\mathbb{E}(X_{n+h} - P_n X_{n+h})^2 = \mathbb{E} \left[X_{n+h} - \left(a_0 + \sum_{i=1}^n a_i X_{n+1-i} \right) \right]^2.$$

We wish to find the values a_0, a_1, \dots, a_n which minimize the above expression.

Let $\mathbf{X} = (X_1, \dots, X_n)^T$ and $\mathbf{a} = (a_1, \dots, a_n)^T$, and let $\tilde{\mathbf{X}}_n = (X_n, \dots, X_1)^T$ be the vector \mathbf{X}_n with the entries in reverse order. In addition, let μ and $\gamma(h)$ be the mean and autocovariance function of $\{X_t, t \in \mathbb{Z}\}$, respectively, and define

$$\mathbf{\Gamma}_n = (\gamma(i-j))_{1 \leq i, j \leq n} \quad \text{and} \quad \boldsymbol{\gamma}_n(h) = (\gamma(h), \gamma(h+1), \dots, \gamma(n+h-1))^T.$$

Then any values of a_0 and \mathbf{a} which minimize the MSPE

$$\mathbb{E}(X_{n+h} - P_n X_{n+h})^2 = \mathbb{E} [X_{n+h} - (a_0 + \mathbf{a}^T \tilde{\mathbf{X}}_n)]^2$$

must satisfy

$$\begin{aligned} a_0 &= \mu(1 - \mathbf{a}^T \mathbf{1}_n) \\ \mathbf{\Gamma}_n \mathbf{a} &= \boldsymbol{\gamma}_n(h). \end{aligned}$$

Moreover, under values of a_0 and \mathbf{a} which satisfy the above equations, the MSPE is given by

$$\mathbb{E}(X_{n+h} - P_n X_{n+h})^2 = \gamma(0) - \mathbf{a}^T \boldsymbol{\gamma}_n(h).$$

Derivation of the above equations

To derive the above equations, we consider predicting the value of any random variable V using the values of some other random variables U_1, \dots, U_n with a predictor of the form $a_0 + \sum_{i=1}^n a_i U_i$. Letting $\mathbf{U} = (U_1, \dots, U_n)^T$ and $\mathbf{a} = (a_1, \dots, a_n)^T$, we consider minimizing the MSPE

$$\mathbb{E}[V - (a_0 + \mathbf{a}^T \mathbf{U})]^2.$$

We find that the values of a_0 and \mathbf{a} which minimize the above expression must satisfy the equations

$$\begin{aligned} a_0 &= \mathbb{E}V - \mathbf{a}^T \mathbb{E}\mathbf{U} \\ \text{Cov}(\mathbf{U})\mathbf{a} &= \text{Cov}(\mathbf{U}, V). \end{aligned}$$

We get the above requirements by taking derivatives of the MSPE with respect to a_0 and \mathbf{a} and setting these to zero. We have

$$\frac{\partial}{\partial a_0} \mathbb{E}[V - (a_0 + \mathbf{a}^T \mathbf{U})]^2 = -2[\mathbb{E}V - a_0 - \mathbf{a}^T \mathbb{E}\mathbf{U}].$$

Setting this equal to zero gives

$$a_0 = \mathbb{E}V - \mathbf{a}^T \mathbb{E}\mathbf{U}.$$

Plugging this value of a_0 into the expression for the MSPE gives

$$\mathbb{E}[V - ((\mathbb{E}V - \mathbf{a}^T \mathbb{E}\mathbf{U}) + \mathbf{a}^T \mathbf{U})]^2 = \mathbb{E}[(V - \mathbb{E}V) - \mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})]^2,$$

and the derivative with respect to \mathbf{a} of the above is

$$\begin{aligned} \frac{\partial}{\partial \mathbf{a}} &= \mathbb{E}[(V - \mathbb{E}V) - \mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})]^2 = -2\mathbb{E}(\mathbf{U} - \mathbb{E}\mathbf{U})[(V - \mathbb{E}V) - \mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})] \\ &= -2 \text{Cov}(\mathbf{U}, V) + 2 \text{Cov}(\mathbf{U})\mathbf{a}. \end{aligned}$$

Setting this equal to zero gives

$$\text{Cov}(\mathbf{U})\mathbf{a} = \text{Cov}(\mathbf{U}, V).$$

We see that if a_0 and \mathbf{a} satisfy the above, then

$$\begin{aligned} \mathbb{E}[V - (a_0 + \mathbf{a}^T \mathbf{U})]^2 &= \mathbb{E}[(V - \mathbb{E}V) - \mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})]^2 \\ &= \mathbb{E}[(V - \mathbb{E}V)^2 - 2\mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})(V - \mathbb{E}V) + \mathbf{a}^T (\mathbf{U} - \mathbb{E}\mathbf{U})(\mathbf{U} - \mathbb{E}\mathbf{U})^T \mathbf{a}] \\ &= \text{Var } V - 2\mathbf{a}^T \text{Cov}(\mathbf{U}, V) + \mathbf{a}^T \text{Cov}(\mathbf{U})\mathbf{a} \\ &= \text{Var } V - \mathbf{a}^T \text{Cov}(\mathbf{U}, V). \end{aligned}$$

Making forecasts

If the matrix $\mathbf{\Gamma}_n$ is non-singular, then the values of a_0 and \mathbf{a} which minimize the MSPE are

$$\mathbf{a} = \mathbf{\Gamma}_n^{-1} \boldsymbol{\gamma}_n(h) \quad \text{with} \quad a_0 = \mu(1 - \mathbf{1}_n^T \mathbf{a}).$$

Example:

Consider forecasting the value of X_{n+h} from the stationary time series defined by

$$X_t = \phi X_{t-1} + Z_t, \quad |\phi| < 1,$$

where $\{Z_t, t \in \mathbb{Z}\}$ is a white noise sequence with mean zero and variance σ^2 . Then the autocovariance function (see Lec 03) is given by

$$\gamma(h) = \frac{\phi^{|h|}}{1 - \phi^2} \sigma^2.$$

Therefore we have

$$\mathbf{\Gamma}_n = \begin{pmatrix} 1 & \phi & \phi^2 & \dots & \phi^{n-2} & \phi^{n-1} \\ \phi & 1 & \phi & \dots & \phi^{n-3} & \phi^{n-2} \\ \phi^2 & \phi & 1 & \dots & \phi^{n-4} & \phi^{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi^{n-2} & \phi^{n-3} & \phi^{n-4} & \dots & 1 & \phi \\ \phi^{n-1} & \phi^{n-2} & \phi^{n-3} & \dots & \phi & 1 \end{pmatrix} \frac{\sigma^2}{1 - \phi^2}$$

and

$$\gamma_n(h) = (\phi^h, \phi^{h+1}, \dots, \phi^{h+n-1}) \frac{\sigma^2}{1 - \phi^2},$$

so that $\mathbf{a} = (\phi^h, 0, \dots, 0)^T$ is the solution to $\mathbf{\Gamma}_n \mathbf{a} = \gamma_n(h)$. The mean of the time series is equal to zero, so $a_0 = 0$. The following R code demonstrates forecasting in the AR(1) model: Note that if we assume a mean-zero model for our time series, we should center the data by subtracting the mean.

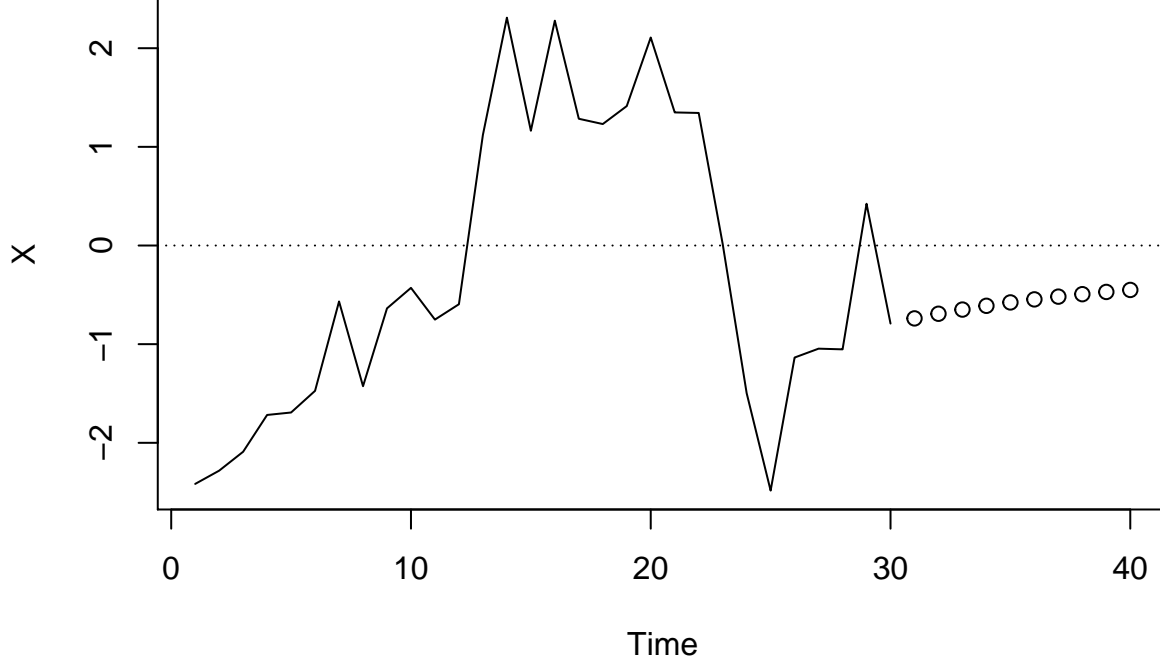
```
phi <- .9
n <- 30
B <- 1000 # length of burn-in period
X0 <- numeric(B+n)
X0[1] <- 0

for( i in 2:(B + n))
{
  X0[i] <- phi * X0[i-1] + rnorm(1)
}

X <- X0[-c(1:B)] # remove burn-in obs from beginning
h <- 10 # forecast h steps ahead

# compute prediction based on centered data, then add the mean back
Xpred <- (X[n]-mean(X)) * phi^c(1:h) + mean(X)

plot(X,xlim=c(1,n+h),xlab="Time",type="l")
points(Xpred~c((n+1):(n+h)))
abline(h=0,lty=3)
```



Fast one-step-ahead forecasting with the Durbin-Levinson algorithm

Consider a stationary time series $\{X_t, t \in \mathbb{Z}\}$ and assume that it has mean $\mu = 0$ and autocovariance function $\gamma(\cdot)$; for the purposes of forecasting, we do not lose any generality by assuming that $\mu = 0$, since we can always work with a centered time series. We now just focus on forecasting the value of X_{n+1} based on the values X_1, \dots, X_n ; that is, we consider the case of one-step-ahead forecasting. For each $n = 1, 2, \dots$, our linear forecaster has the form

$$P_n X_{n+1} = \sum_{i=1}^n a_{n,i} X_{n+1-i} = \mathbf{a}_n^T \tilde{\mathbf{X}}_n,$$

where $\mathbf{a}_n = (a_{1,1}, \dots, a_{n,n})^T$. The choice of \mathbf{a}_n which minimizes the MSPE must satisfy the equation

$$\mathbf{\Gamma}_n \mathbf{a}_n = \boldsymbol{\gamma}_n, \quad \text{where} \quad \boldsymbol{\gamma}_n := \boldsymbol{\gamma}_n(n) = (\gamma(1), \dots, \gamma(n))^T.$$

If the matrix $\mathbf{\Gamma}_n$ is non-singular, then the solution is unique and is given by

$$\mathbf{a}_n = \mathbf{\Gamma}_n^{-1} \boldsymbol{\gamma}_n.$$

If n is very large, then computing the inverse of the $n \times n$ matrix $\mathbf{\Gamma}_n$ could be very computationally expensive. In the following we derive the Durbin-Levinson algorithm, which is an algorithm to recursively compute one-step-ahead forecasts $P_n X_{n+1}$ for each $n \geq 1$ without having to invert the matrices $\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \dots$

For each $k \geq 1$, define $\mathbf{a}_k = (a_{k,1}, \dots, a_{k,k})^T$ by

$$\mathbf{a}_k = \mathbf{\Gamma}_k^{-1} \boldsymbol{\gamma}_k,$$

where $\boldsymbol{\gamma}_k = (\gamma(1), \dots, \gamma(k))^T$. In addition define $\tilde{\mathbf{a}}_k = (a_{k,k}, \dots, a_{k,1})^T$ and $\tilde{\boldsymbol{\gamma}}_k = (\gamma(k), \dots, \gamma(1))^T$, which are the vectors \mathbf{a}_k and $\boldsymbol{\gamma}_k$ with the entries in reverse order. We wish to find an expression for the vector \mathbf{a}_{k+1} in terms of \mathbf{a}_k which does not require an inversion of the matrix $\mathbf{\Gamma}_{k+1}$. We begin by writing

$$\begin{aligned}
\begin{bmatrix} a_{k+1,1} \\ \vdots \\ a_{k+1,k+1} \end{bmatrix} &= \mathbf{\Gamma}_{k+1} \boldsymbol{\gamma}_{k+1} \\
&= \begin{bmatrix} \mathbf{\Gamma}_k & \tilde{\boldsymbol{\gamma}}_k \\ \tilde{\boldsymbol{\gamma}}_k^T & \gamma(0) \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\gamma}_k \\ \vdots \\ \gamma(k+1) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{\Gamma}_k^{-1} + \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} & -\mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \\ -\tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} & (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma}_k \\ \vdots \\ \gamma(k+1) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{\Gamma}_k^{-1} \boldsymbol{\gamma}_k + \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \boldsymbol{\gamma}_k - \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \gamma(k+1) \\ -\tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \boldsymbol{\gamma}_k + (\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k)^{-1} \gamma(k+1) \end{bmatrix}
\end{aligned}$$

Replacing $\mathbf{\Gamma}_k^{-1} \boldsymbol{\gamma}_k$ with \mathbf{a}_k and $\mathbf{\Gamma}_k^{-1} \tilde{\boldsymbol{\gamma}}_k$ with $\tilde{\mathbf{a}}_k$, we have

$$\begin{bmatrix} a_{k+1,1} \\ \vdots \\ a_{k+1,k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_k - \frac{\gamma(k+1) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{a}_k}{\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \tilde{\mathbf{a}}_k} \tilde{\mathbf{a}}_k \\ \frac{\gamma(k+1) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{a}_k}{\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \tilde{\mathbf{a}}_k} \end{bmatrix}.$$

From this we see that we can write

$$\begin{aligned}
a_{k+1,k+1} &= \frac{\gamma(k+1) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{a}_k}{\gamma(0) - \tilde{\boldsymbol{\gamma}}_k^T \mathbf{a}_k} \\
(a_{k+1,1}, \dots, a_{k+1,k})^T &= \mathbf{a}_k - a_{k+1,k+1} \tilde{\mathbf{a}}_k,
\end{aligned}$$

noting that $\tilde{\boldsymbol{\gamma}}_k^T \tilde{\mathbf{a}}_k = \boldsymbol{\gamma}_k^T \mathbf{a}_k$. With this recursive algorithm we can find \mathbf{a}_{k+1} from \mathbf{a}_k without having to invert the matrix $\mathbf{\Gamma}_{k+1}$. We can make one further tweak so that computation is even faster: Set $v_k = \gamma(0) - \boldsymbol{\gamma}_k^T \mathbf{a}_k$ for $k \geq 1$ and $v_0 = \gamma(0)$. Then we have

$$\begin{aligned}
v_k &= \gamma(0) - \boldsymbol{\gamma}_k^T \mathbf{a}_k \\
&= \gamma(0) - \gamma(k) a_{k,k} - \boldsymbol{\gamma}_{k-1}^T (a_{k,1}, \dots, a_{k,k-1})^T \\
&= \gamma(0) - \gamma(k) a_{k,k} - \boldsymbol{\gamma}_{k-1}^T (\mathbf{a}_{k-1} - a_{k,k} \tilde{\mathbf{a}}_{k-1}) \\
&= v_{k-1} - a_{k,k} (\gamma(k) - \boldsymbol{\gamma}_{k-1}^T \tilde{\mathbf{a}}_{k-1}) \\
&= v_{k-1} - a_{k,k} \frac{(\gamma(k) - \boldsymbol{\gamma}_{k-1}^T \tilde{\mathbf{a}}_{k-1})}{\gamma(0) - \boldsymbol{\gamma}_{k-1}^T \mathbf{a}_{k-1}} (\gamma(0) - \boldsymbol{\gamma}_{k-1}^T \mathbf{a}_{k-1}) \\
&= v_{k-1} - a_{k,k}^2 v_{k-1} \\
&= v_{k-1} (1 - a_{k,k}^2).
\end{aligned}$$

This allows us to define the algorithm as described in the following section.

Recursive computation of coefficients with the Durbin-Levinson algorithm:

Set

$$v_0 = \gamma(0) \quad \text{and} \quad a_{1,1} = \gamma(1)/\gamma(0),$$

and then for $k = 1, \dots, n$ perform the recursions

$$\begin{aligned} v_k &= v_{k-1}(1 - a_{k,k}^2) \\ a_{k+1,k+1} &= (\gamma(k+1) - \tilde{\gamma}_k^T \mathbf{a}_k) / v_k \\ (a_{k+1,1}, \dots, a_{k+1,k})^T &= \mathbf{a}_k - a_{k+1,k+1} \tilde{\mathbf{a}}_k. \end{aligned}$$

The one-step-ahead predictions are then computed as

$$P_k X_{k+1} = \begin{cases} 0 & \text{if } k = 0 \\ \mathbf{a}_k^T \tilde{\mathbf{X}}_k & \text{if } k = 1, \dots, n, \end{cases}$$

where $\tilde{\mathbf{X}}_k = (X_k, \dots, X_1)^T$ for $k = 1, \dots, n$.

Note: We obtain $v_0 = \gamma(0)$ by considering the case $k = 1$. Also, the values v_0, v_1, \dots, v_n are the MSPEs for the predictions.

The following R code defines a function for running the Durbin-Levinson algorithm and applies it to a simulated data set from an MA(q) model:

```
DL.1step <- function(X,gamma.0,gamma.n){

  n <- length(X)
  X.pred <- numeric(n+1)
  X.pred[1] <- 0
  alpha <- numeric(n)
  v <- numeric(n+1)
  v[1] <- gamma.0
  a.k <- gamma.n[1] / gamma.0
  alpha[1] <- a.k

  for(k in 1:(n-1))
  {

    X.pred[k+1] <- sum( a.k * X[k:1] )

    v[k+1] <- v[k]*(1 - a.k[k]^2)
    a.kplus1 <- numeric(k+1)

    a.kplus1[k+1] <- ( gamma.n[k+1] - sum( gamma.n[k:1] * a.k ) ) / v[k+1]
    a.kplus1[1:k] <- a.k - a.kplus1[k+1] * a.k[k:1]

    a.k <- a.kplus1

    alpha[k+1] <- a.kplus1[k+1]

  }

  X.pred[n+1] <- sum( a.k * X[n:1] )
```

```

    output <- list( X.pred = X.pred,
                   alpha = alpha,
                   v = v)

    return(output)
}

# generate some data from an MA(q) model
n <- 100
theta <- c(1,.8,.6,.5,.25,.1,.1,.95)
q <- length(theta) - 1
Z <- rnorm(n+q,0,1)
X <- numeric(n)

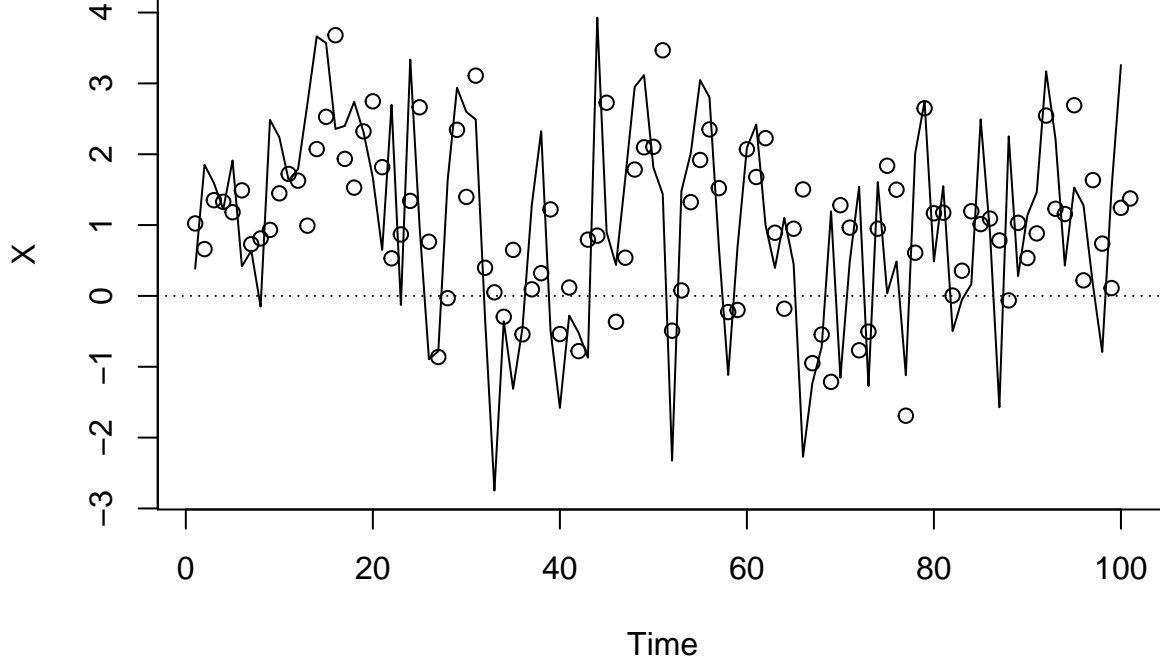
for( t in 1:n)
{
    ind <- q + t:(t-q)
    X[t] <- sum( theta * Z[ind] )
}

# construct vector of autocovariances to use in the DL algorithm
gamma.0 <- 1 * sum(theta^2)
gamma.n <- numeric(n)
for( h in 1:q )
{
    gamma.n[h] <- sum(theta[1:(q - h + 1)]*theta[ ( 1 + h ) : ( q + 1 ) ])
}

# run the Durbin-Levinson algorithm in centered data, then add center back
X.pred <- DL.1step(X-mean(X),gamma.0,gamma.n)$X.pred + mean(X)

# plot original series as well as one-step-ahead predictions
plot(X,xlim=c(1,n+1),xlab="Time",type="l",ylim=range(X,X.pred))
points(X.pred)
abline(h=0,lty=3)

```



Fast one-step- and h-step-ahead forecasting with the innovations algorithm

Another algorithm with which we can compute one-step-ahead forecasts without inverting large matrices is the so-called innovations algorithm. This algorithm does not require stationarity, though it assumes a constant mean. In this section we consider a time series $\{X_t, t \in \mathbb{Z}\}$ such that $\mathbb{E}X_t = 0$ for all $t \in \mathbb{Z}$, we and define $\kappa(i, j) = \mathbb{E}X_i X_j$ for $i, j \in \mathbb{Z}$.

For $k = 1, 2, \dots$ define the matrices $\mathbf{K}_k = (\kappa(i - j))_{1 \leq i, j \leq k}$, and assume that these are nonsingular matrices. Then define the one-step-ahead predictions $\hat{X}_1, \dots, \hat{X}_n$ of X_1, \dots, X_n as

$$\hat{X}_{k+1} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=1}^k a_{k,j} X_{k+1-j} & \text{if } k = 1, \dots, n-1, \end{cases}$$

where $\mathbf{a}_k = (a_{k,1}, \dots, a_{k,k})^T$ is given by

$$\mathbf{a}_k = \underset{\mathbf{a} \in \mathbb{R}^k}{\operatorname{argmin}} \mathbb{E}(X_{k+1} - \mathbf{a}^T \tilde{\mathbf{X}}_k)^2 = \mathbf{K}_n^{-1}(\kappa(k, k+1), \dots, \kappa(1, k+1))^T$$

for $k = 1, \dots, n-1$. Define the vector $\hat{\mathbf{X}}_n = (\hat{X}_1, \dots, \hat{X}_n)^T$ and let $\mathbf{U}_n = \hat{\mathbf{X}}_n - \mathbf{X}_n$, so that \mathbf{U} contains the one-step-ahead prediction errors. We can call these the “innovations”. Define the matrix

$$\mathbf{A}_n = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ a_{1,1} & 0 & 0 & \dots & 0 & 0 \\ a_{2,2} & a_{2,1} & 0 & \dots & 0 & 0 \\ a_{3,3} & a_{3,2} & a_{3,1} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ a_{n-1,n-1} & a_{n-1,n-2} & a_{n-1,n-3} & \dots & a_{n-1,1} & 0 \end{bmatrix},$$

so that $\hat{\mathbf{X}}_n = \mathbf{A}_n \mathbf{X}_n$. Then we see that we may write

$$\mathbf{U}_n = (\mathbf{I} - \mathbf{A}_n) \mathbf{X}_n.$$

Moreover, since $(\mathbf{I} - \mathbf{A}_n)$ is non-singular, we may write

$$(\mathbf{I} - \mathbf{A}_n)^{-1} \mathbf{U}_n = \mathbf{X}_n,$$

where $(\mathbf{I} - \mathbf{A}_n)^{-1} = (\mathbf{I} + \mathbf{\Theta}_n)$, where the matrix $\mathbf{\Theta}_n$ has the form

$$\mathbf{\Theta}_n = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ \theta_{1,1} & 0 & 0 & \dots & 0 & 0 \\ \theta_{2,2} & \theta_{2,1} & 0 & \dots & 0 & 0 \\ \theta_{3,3} & \theta_{3,2} & \theta_{3,1} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ \theta_{n-1,n-1} & \theta_{n-1,n-2} & \theta_{n-1,n-3} & \dots & \theta_{n-1,1} & 0 \end{bmatrix}.$$

Then we may write

$$\begin{aligned} \hat{\mathbf{X}}_n &= \mathbf{X}_n - (\mathbf{I} - \mathbf{A}_n) \mathbf{X}_n \\ &= (\mathbf{I} - \mathbf{A}_n)^{-1} \mathbf{U}_n - \mathbf{U}_n \\ &= (\mathbf{I} + \mathbf{\Theta}_n) \mathbf{U}_n - \mathbf{U}_n \\ &= \mathbf{\Theta}_n \mathbf{U}_n. \end{aligned}$$

This gives

$$\hat{X}_{k+1} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=1}^k \theta_{k,j} (X_{k+1-j} - \hat{X}_{k+1-j}) & \text{if } k = 1, \dots, n-1. \end{cases}$$

We find that we can compute the values $\theta_{n,j}$, $j = 1, \dots, n$, $n = 1, 2, \dots$ recursively, allowing fast computation of the one-step-ahead predictions. This is described in the following section.

Recursive computation of coefficients with the innovations algorithm:

Set $v_0 = \kappa(1, 1)$. Then for $k = 1, \dots, n-1$, set

$$\theta_{k,k} = v_0^{-1} \kappa(k+1, 1)$$

and then

$$\theta_{k,k-j} = v_j \left[\kappa(k+1, j+1) - \sum_{l=0}^{j-1} \theta_{j,j-l} \theta_{k,k-l} v_l \right], \quad \text{for } j = 1, \dots, k-1$$

and

$$v_k = \kappa(k+1, k+1) - \sum_{l=0}^{k-1} \theta_{k,k-l}^2 v_l.$$

Moreover, we find that we can get the h -step-ahead predictions \hat{X}_{n+h} , $h = 1, 2, \dots$ according to

$$\hat{X}_{n+h} = \sum_{j=h}^{n+h-1} \theta_{n+h-1,j} (X_{n+h-j} - \hat{X}_{n+h-j}),$$

where the values of $\theta_{n+h-1,j}$, $j = h, \dots, n+h-1$, $h = 1, 2, \dots$ can be found by continuing the recursions, but with

$$v_{n+h} = \kappa(n+h, n+h) - \sum_{j=h}^{n+h-1} \theta_{n+h-1,j}^2 v_{n+h-j}$$

(see pg. 167 of B&D Theory). As with the Durbin-Levinson algorithm, the values $v_0, \dots, v_n, v_{n+1}, \dots, v_{n+h}$ are the MSPEs for the predictions.

The following R code implements h -step ahead forecasting with the innovations algorithm:

```
# build a function to perform the innovations algorithm
innov.hstep<- function(X,h,K){
  #
  # X = vector of data values
  # h = number of step ahead for which to predict
  # K = covariance matrix of X_1,dots,X_{n+h}
  #

  n <- length(X)
  v <- numeric(n+h)
  X.pred <- numeric(n+h)
  Theta <- matrix(NA,n+h,n+h)

  v[1] <- K[1,1]
  X.pred[1] <- 0

  Theta[1,1] <- K[2,1] / v[1]
  v[2] <- K[2,2] - Theta[1,1]^2*v[1]
  X.pred[2] <- Theta[1,1]*X[1]

  for(k in 2:n)
  {

    Theta[k,k] <- K[k+1,1] / v[1]

    for(j in 1:(k-1))
    {

      Theta[k,k-j] <- (K[k+1,j+1]-sum(Theta[j,j:1]*Theta[k,k:(k-j+1)]*v[1:j]))/v[j+1]

    }

    v[k+1] <- K[k+1,k+1] - sum( Theta[k,k:1]^2 * v[1:k] )
    X.pred[k+1] <- sum( Theta[k,1:k] *(X[k:1] - X.pred[k:1]) )

  }

  for(k in (n+1):(n+h-1))
  {

    Theta[k,k] <- K[k+1,1] / v[1]

    for(j in 1:(k-1))
    {
```

```

        Theta[k,k-j]<-(K[k+1,j+1]-sum(Theta[j,j:1]*Theta[k,k:(k-j+1)]*v[1:j]))/v[j+1]
    }

    v[k+1] <- K[k+1,k+1] - sum( Theta[k,(k-n+1):k]^2 * v[n:1] )
    X.pred[k+1] <- sum( Theta[k,(k-n+1):k] *(X[n:1] - X.pred[n:1]) )

}

output <- list( X.pred = X.pred,
               v = v)

return(output)
}

# generate some data from an MA(q) model
n <- 100
theta <- c(1,.8,.6,.5,.25,.1,.1,.95)
q <- length(theta) - 1
Z <- rnorm(n+q,0,1)
X <- numeric(n)

for( t in 1:n)
{
    ind <- q + t:(t-q)
    X[t] <- sum( theta * Z[ind] )
}

# construct covariance matrix K
h <- 10
gamma.0 <- 1 * sum(theta^2)
gamma.nplush <- numeric(n+h)
for( l in 1:q )
{
    gamma.nplush[l] <- sum(theta[1:(q - l + 1)]*theta[ ( 1 + l ) : ( q + 1 ) ])
}

K <- matrix(NA,n+h,n+h)
for(j in 1:(n+h))
    for(i in 1:(n+h))
    {

        K[i,j] <- c(gamma.0,gamma.nplush)[1+abs(i-j)]

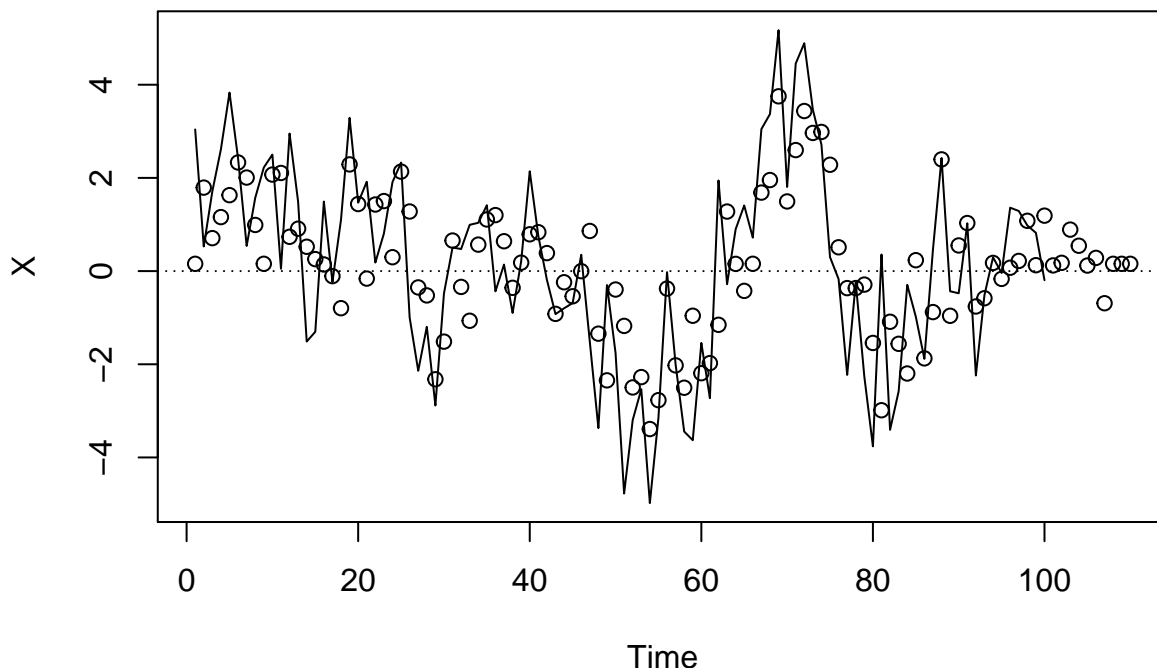
    }

# run the innovations algorithm in centered data, then add mean back
X.pred <- innov.hstep(X-mean(X),h,K)$X.pred + mean(X)

# plot original series as well as predictions
plot(X,xlim=c(1,n+h),xlab="Time",ylim=range(X,X.pred),type="l")
points(X.pred)

```

```
abline(h=0,lty=3)
```



The partial autocorrelation function

In addition to the autocorrelation function, another function, called the *partial autocorrelation function* (pacf), carries information about the dependence structure of a stationary time series. For a stationary time series $\{X_t, t \in \mathbb{Z}\}$ with mean 0 and any random variable Y , define

$$P_{\overline{\text{sp}}\{X_2, \dots, X_k\}} Y = \sum_{j=2}^k a_j X_j,$$

where a_2, \dots, a_k are the values which minimize the expression

$$\mathbb{E} \left(Y - \sum_{j=2}^k a_j X_j \right)^2.$$

So $P_{\overline{\text{sp}}\{X_2, \dots, X_k\}} Y$ is the projection of Y onto the space spanned by X_2, \dots, X_k . The partial autocorrelation function $\alpha(\cdot)$ is given by

$$\alpha(k) = \text{Corr}(X_{k+1} - P_{\overline{\text{sp}}\{X_2, \dots, X_k\}} X_{k+1}, X_1 - P_{\overline{\text{sp}}\{X_2, \dots, X_k\}} X_1)$$

for $k = 1, 2, \dots$. We can interpret $\alpha(k)$ as the correlation between X_{k+1} and X_1 after accounting for the effects of the intermediate random variables X_2, \dots, X_k . That is, if we were to regress both X_{k+1} and X_1 onto X_2, \dots, X_k and get the residuals, the correlation between these residuals would be equal to $\alpha(k)$.

Interestingly, it turns out that the Durbin-Levinson algorithm computes the partial autocorrelations $\alpha(1), \dots, \alpha(n)$ as it goes through its recursions. In fact, the partial autocorrelations are equal to

$$\alpha(k) = a_{k,k}, \quad \text{for } k = 1, \dots, n.$$

We will discuss the pacf more later on.

Building prediction intervals for Gaussian processes

For a stationary time series $\{X_t, t \in \mathbb{Z}\}$ in which each X_t has a Normal distribution, prediction intervals can be constructed using quantiles from the Normal distribution according to

$$\hat{X}_{k+1} \pm z_\alpha \sqrt{v_k}, \quad k = 1, \dots, n + h - 1,$$

where z_α is the upper α quantile of the standard Normal distribution and where the MSPEs v_1, \dots, v_{n+h-1} can be obtained from the innovations algorithm. The R code below implements this on the data set from in the previous chunk of R code.

```
innov.hstep.out <- innov.hstep(X-mean(X),h,K)
X.pred <- innov.hstep.out$X.pred + mean(X)
v <- innov.hstep.out$v

lo.pred <- X.pred - 1.96 * sqrt(v)
up.pred <- X.pred + 1.96 * sqrt(v)

# plot original series as well as predictions with prediction limits
plot(X,xlim=c(n-10,n+h),xlab="Time",type="l",ylim=range(X,lo.pred,up.pred))
points(X.pred)
lines(lo.pred,lty=3)
lines(up.pred,lty=3)
abline(h=0,lty=3)
```

