hw4code

Han-Yuan Hsu

2022-10-26

```
library(tidyverse)
set.seed(111)
```

Reference: lecture code

1

Consider the dataset lynx that is available in base R. This gives the annual numbers of lynx trappings for 1821-1934 in Canada. Type help(lynx) to learn more about the dataset.

```
data("lynx")
n = length(lynx) # 114
```

a

Fit the AR(2) model to the first 90 observations of this dataset. Report the estimates of $\phi_0, \phi_1, \phi_2, \sigma$ along with uncertainty quantification. (3 points)

```
p = 2
Xmat = matrix(1, (90-p), 1)
for(j in 1:p)
{
   Xmat = cbind(Xmat, lynx[(p+1-j):(90-j)])
}
modar = lm(lynx[(p+1):90] ~ -1 + Xmat)
summary(modar)
```

```
##
## Call:
## lm(formula = lynx[(p + 1):90] ~ -1 + Xmat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2208.5  -499.4  -223.1   334.3  3178.4
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## Xmat1 697.54824  136.77280   5.100 2.03e-06 ***
```

```
## Xmat2   1.16477    0.08475  13.744  < 2e-16 ***
## Xmat3  -0.62321    0.08470  -7.358 1.09e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 914.3 on 85 degrees of freedom
## Multiple R-squared:  0.8394, Adjusted R-squared:  0.8337
## F-statistic: 148.1 on 3 and 85 DF,  p-value: < 2.2e-16
```

The point estimates for ϕ_0, ϕ_1, ϕ_2 are

```
modar$coefficients
```

```
##       Xmat1       Xmat2       Xmat3
## 697.5482377   1.1647679  -0.6232059
```

The point estimate for σ is

```
sighat = sqrt((sum((modar$residuals)^2))/85) # 85=90-2*2-1
sighat
```

```
## [1] 914.2592
```

`sighat` is quite large. A possible reason can be seen in the plot in part (d). It reveals that the AR(2) model cannot fit the huge peaks present every 40 years in the data, and the model treats these huge peaks as variances of the data.

Uncertainty quantification:
The posterior phi's follow a multivariate t distribution.

```
cov.mat = sighat^2 * solve(t(Xmat) %*% Xmat)
sd = sqrt(diag(cov.mat))
sd
```

```
## [1] 136.77279532   0.08474811   0.08470104
```

The rows below are 95% confidence intervals for ϕ_0, ϕ_1, ϕ_2:

```
for(i in 1:3) {
  a = as.numeric(modar$coefficients[i] + sd[i]*qt(p=.025, df=90-2*2-1))
  b = as.numeric(modar$coefficients[i] + sd[i]*qt(p=.975, df=90-2*2-1))
  print(c(a,b))
}
```

```
## [1] 425.6073 969.4892
## [1] 0.9962659 1.3332698
## [1] -0.7916142 -0.4547975
```

From homework 1, we know the posterior σ is such that

$$\frac{\text{RSS}}{\sigma^2}$$

follows the chi-square distribution with 90-2*2-1 degrees of freedom. Below is a 95% CI for σ:

```r
chi975 = qchisq(.975, df=90-5)
chi025 = qchisq(.025, df=90-5)
RSS = sum((modar$residuals)^2)
c(sqrt(RSS/chi975), sqrt(RSS/chi025))
```

```
## [1]  795.0756 1075.8068
```

b

Write down an explicit formula for the predictions generated by your fitted AR(2) model for Yt for t $\geq$ 91. (4 points)

The explicit formula is of the form `c + r^t * (a*cos(theta*t) + b*sin(theta*t))`, where c, r, a, b, and theta will be calculated in the following code blocks.

```
phi0 = modar$coefficients[1]
phi1 = modar$coefficients[2]
phi2 = modar$coefficients[3]
root = polyroot(c(-phi2, -phi1, 1))[1]
r = Mod(root)
theta = Arg(root)
r
```

```
## [1] 0.7894339
```

```
theta
```

```
## [1] 0.7411043
```

```
c = as.numeric(phi0 / (1-phi1-phi2))
c
```

```
## [1] 1521.576
```

```
a = lynx[89] - c
b = ((lynx[90]-c)/r - a*cos(theta))/sin(theta)
a
```

```
## [1] -1139.576
```

```
b
```

```
## [1] -93.63978
```

Below is the explicit formula for the predicted y_t, where $t \geq 91$:

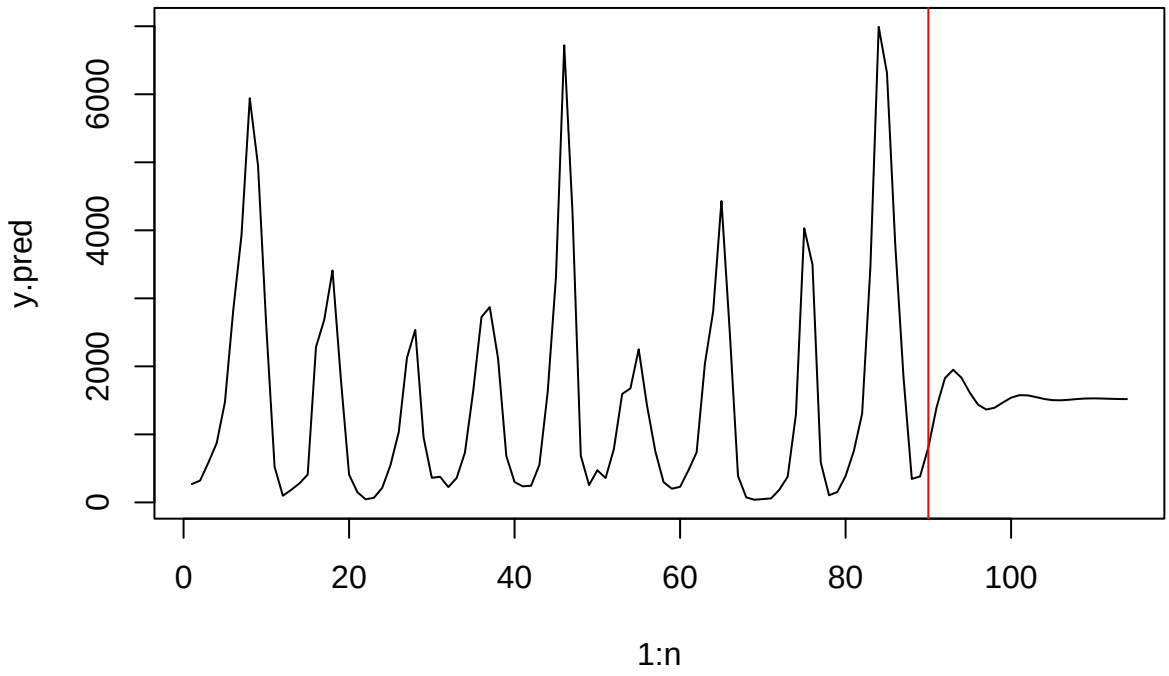
```
pred = function(t) {
  t = t-89
  return( c + r^t * (a*cos(theta*t) + b*sin(theta*t)) )
}
```

c

Use your AR(2) to predict the data from time points t = 91, . . . , 114. Also compute the standard deviations corresponding to the accuracy of prediction. (4 points).

In the plot below, predictions are on the right side of the red line.

```
y.pred = rep(-9999, n)
y.pred[1:90] = lynx[1:90]
for (k in 91:n) {
  y.pred[k] = sum(modar$coefficients * c(1, y.pred[k-1], y.pred[k-2]))
}

plot(1:n, y.pred, type='l')
abline(v=90, col='red')
```




```
#lines(89:n, sapply(89:n, pred), col='blue')
```

Calculate covariance matrix of prediction variables, Gamhat:

```
Gamhat = matrix(sighat^2, 1, 1) #this is the uncertainty for the first i.e., (n+1)^th prediction
vkp = matrix(modar$coefficients[2], 1, 1) #this is the estimate for phi1
for(i in 1:(n-90-1))
{
    covterm = Gamhat %*% vkp
    varterm = (sighat^2) + (t(vkp) %*% (Gamhat %*% vkp))
    Gamhat = cbind(Gamhat, covterm)
    Gamhat = rbind(Gamhat, c(t(covterm), varterm))

    if (i < p) {vkp = c(modar$coefficients[(i+2)], vkp)}
    if (i >= p) {vkp = c(0, vkp)}
}
```

The standard deviations corresponding to the accuracy of prediction:

```
predsd = sqrt(diag(Gamhat))
predsd
```

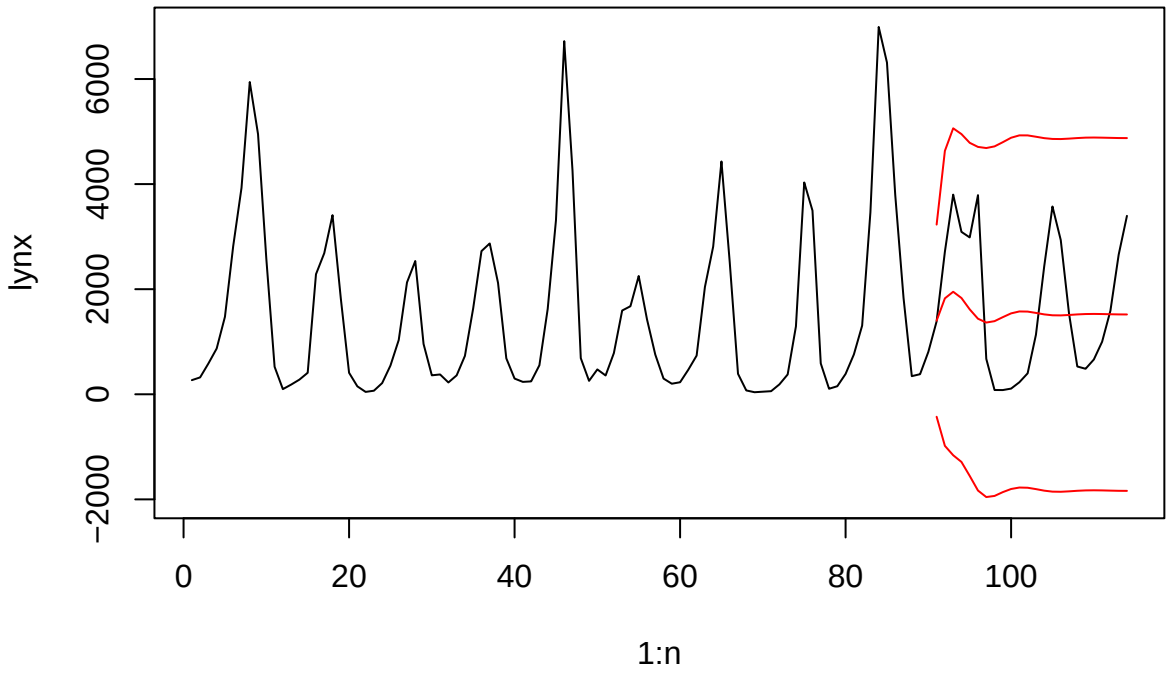
```
##  [1]  914.2592 1403.5246 1555.4971 1559.9234 1585.0549 1634.9047 1660.6794
##  [8] 1663.1049 1664.8888 1671.2522 1675.6768 1676.4509 1676.5321 1677.3068
## [15] 1678.0248 1678.2171 1678.2174 1678.3034 1678.4130 1678.4540 1678.4549
## [22] 1678.4633 1678.4790 1678.4869
```

d

Compare your predictions with the actual values from the dataset. Comment on the accuracy of the predictions. (2 points)

In the plot below, the black graph is the actual lynx data, the red graphs are the predictions plus or minus 2 standard deviations.

```
plot(1:n, lynx, type='l', ylim=c(-2000, 7000))
lines(91:n, y.pred[91:n], col='red')
predlower = y.pred[91:n] - 2*predsd
predupper = y.pred[91:n] + 2*predsd
lines(91:n, predlower, col='red')
lines(91:n, predupper, col='red')
```



The predictions are not accurate; they quickly converge to one value, while the actual data continues to

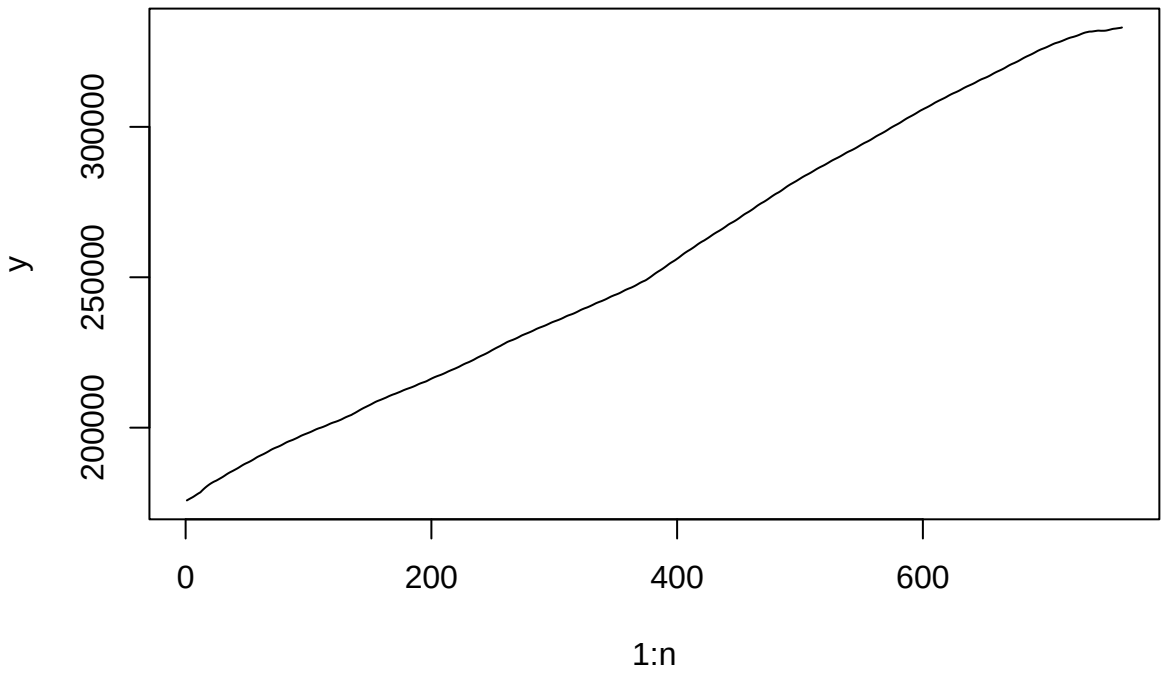
have seasonal oscillations. The model just treats those oscillations as errors and so the standard deviation is high. That is why the actual data is contained in the 2-sd band.

2

Consider the US population dataset from https://fred.stlouisfed.org/series/ POPTHM that we have worked with in class.

```
pop = read.csv('POPTHM.csv')
y = pop$POPTHM
n = length(y)
```

```
plot(1:n, y, type='l')
```



a

Fit an AR(2) model to this dataset. Report the estimates of the parameters along with uncertainty quantification (3 points).

```
Xmat = matrix(1, nrow=n-2, ncol=3)
Xmat[,2] = y[2:(n-1)]
Xmat[,3] = y[1:(n-2)]
modar = lm(y[3:n] ~ -1 + Xmat)
summary(modar)
```

```
##
```

```
## Call:
## lm(formula = y[3:n] ~ -1 + Xmat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -64.649 -12.007   1.506  10.827 190.367
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## Xmat1 12.98925    4.36137   2.978  0.00299 **
## Xmat2  1.95181    0.01137 171.640  < 2e-16 ***
## Xmat3 -0.95182    0.01137 -83.715  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.75 on 757 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 6.131e+10 on 3 and 757 DF,  p-value: < 2.2e-16
```

The point estimates for ϕ_0, ϕ_1, ϕ_2 are

```
modar$coefficients
```

```
##      Xmat1      Xmat2      Xmat3
## 12.9892477  1.9518065 -0.9518189
```

The point estimate for σ is

```
sighat = sqrt((sum((modar$residuals)^2))/(n-2*2-1))
sighat
```

```
## [1] 16.75288
```

Uncertainty quantification:
The posterior phi's follow a multivariate t distribution.

```
cov.mat = sighat^2 * solve(t(Xmat) %*% Xmat)
sd = sqrt(diag(cov.mat))
sd
```

```
## [1] 4.36136883 0.01137148 0.01136977
```

The rows below are 95% confidence intervals for ϕ_0, ϕ_1, ϕ_2:

```
for(i in 1:3) {
  a = as.numeric(modar$coefficients[i] + sd[i]*qt(p=.025, df=n-2*2-1))
  b = as.numeric(modar$coefficients[i] + sd[i]*qt(p=.975, df=n-2*2-1))
  print(c(a,b))
}
```

```
## [1]  4.427433 21.551063
## [1] 1.929483 1.974130
## [1] -0.9741390 -0.9294989
```

Below is a 95% CI for σ:

```
chi975 = qchisq(.975, df=n-5)
chi025 = qchisq(.025, df=n-5)
RSS = sum((modar$residuals)^2)
c(sqrt(RSS/chi975), sqrt(RSS/chi025))
```

```
## [1] 15.94990 17.64162
```

b

Write down an explicit formula for the predictions generated by your fitted AR(2) model for Yt for the future months. (4 points)

```
phi0 = modar$coefficients[1]
phi1 = modar$coefficients[2]
phi2 = modar$coefficients[3]
roots = polyroot(c(-phi2, -phi1, 1)) # real roots
roots
```

```
## [1] 0.9520667-0i 0.9997397+0i
```

```
alpha = Re(roots[1]) # convert to real numbers
beta = Re(roots[2])
```

```
c = as.numeric(phi0 / (1-phi1-phi2))
c
```

```
## [1] 1041218
```

```
a = (beta*(y[n-1]-c) - (y[n]-c)) / (beta-alpha)
b = (-alpha*(y[n-1]-c) + (y[n]-c)) / (beta-alpha)
a
```

```
## [1] 1769.107
```

```
b
```

```
## [1] -710059.5
```

Below is the explicit formula for the predicted y_t, where $t > n$:

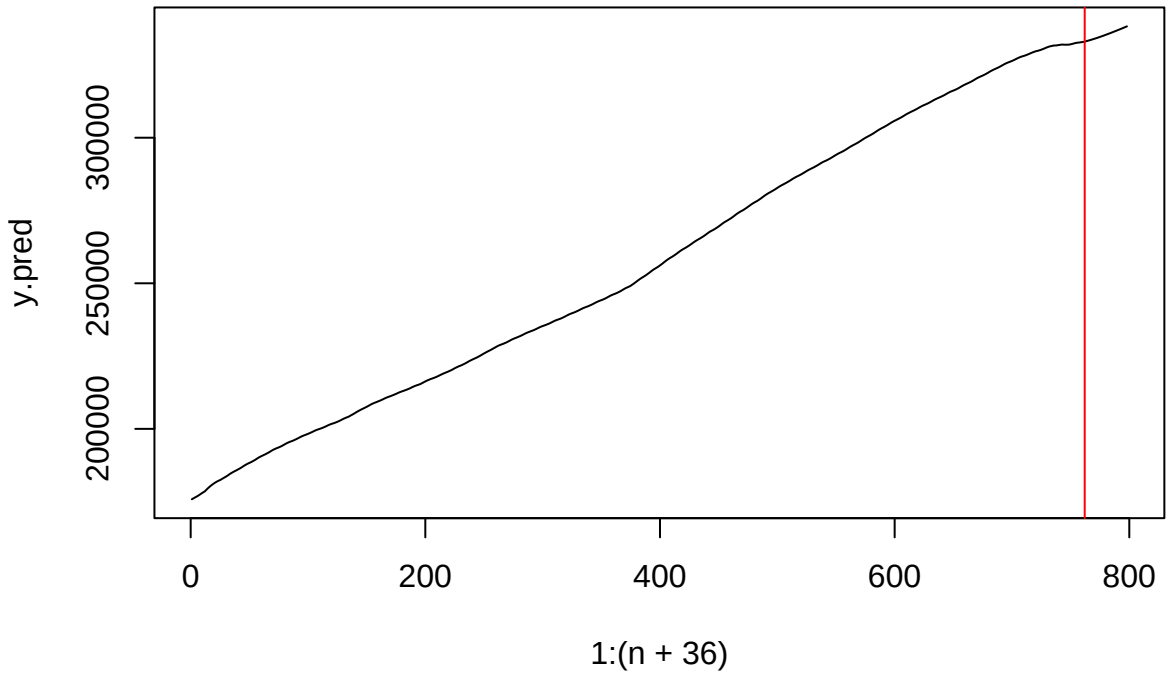
```
pred <- function(t) {
  t = t-(n-1)
  c + a*alpha^t + b*beta^t
}
```

c

Use your AR(2) model to predict the data for 36 months immediately succeeding the last month in the dataset. Plot these predictions and uncertainty indicators along with the original data. Do these predictions make intuitive sense? (6 points)

The prediction starts after the red line; the part on the left side of the red line is the original data.

```
y.pred = c(y, sapply((n+1):(n+36), pred))
plot(1:(n+36), y.pred, type='l')
abline(v=n, col='red')
```




```
#lines(89:n, sapply(89:n, pred), col='blue')
```

Calculate covariance matrix of prediction variables, Gamhat:

```
Gamhat = matrix(sighat^2, 1, 1) #this is the uncertainty for the first i.e., (n+1)^th prediction
vkp = matrix(modar$coefficients[2], 1, 1) #this is the estimate for phi1
p = 2
for(i in 1:(36-1))
{
    covterm = Gamhat %*% vkp
    varterm = (sighat^2) + (t(vkp) %*% (Gamhat %*% vkp))
    Gamhat = cbind(Gamhat, covterm)
    Gamhat = rbind(Gamhat, c(t(covterm), varterm))
```

```
    # update vkp
    if (i < p) {vkp = c(modar$coefficients[(i+2)], vkp)}
    if (i >= p) {vkp = c(0, vkp)}
}
```

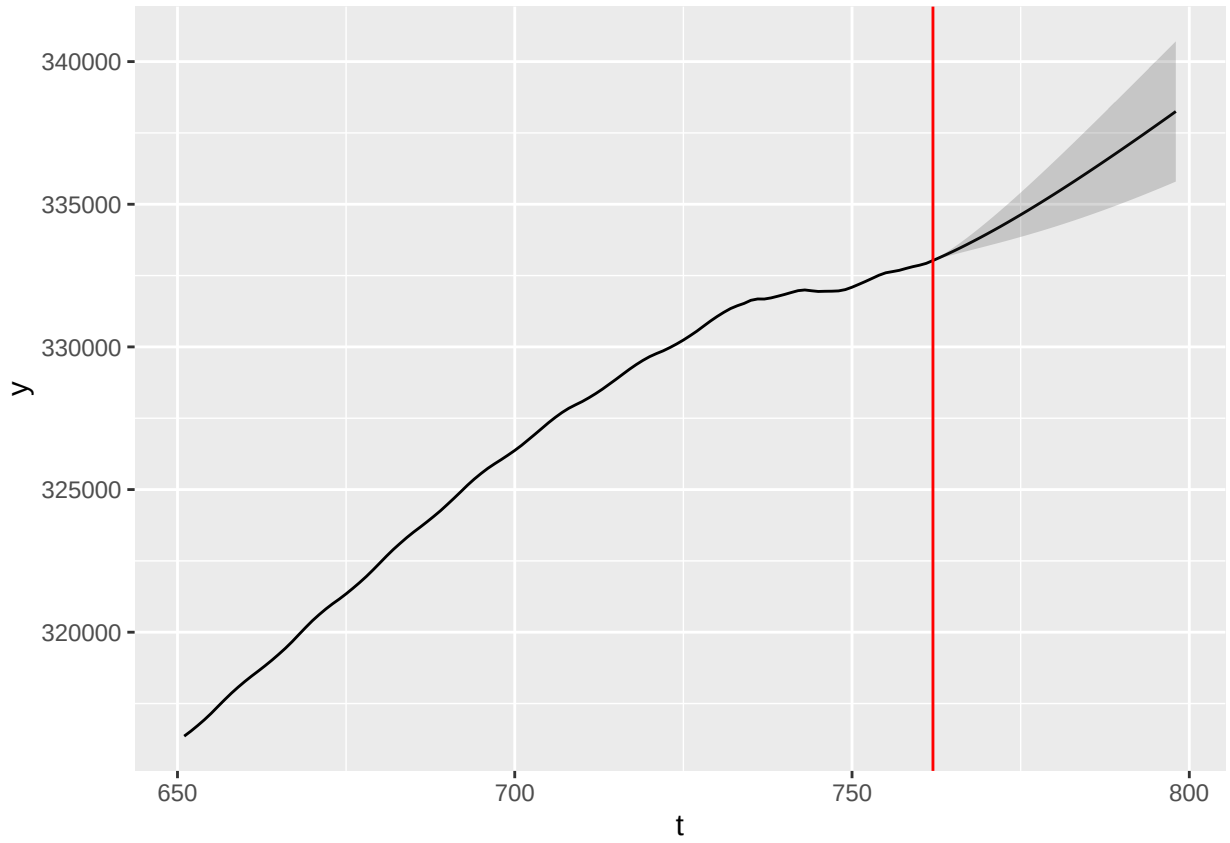
The standard deviations corresponding to the accuracy of prediction:

```
predsd = sqrt(diag(Gamhat))
predsd
```

```
##  [1]   16.75288   36.74020   60.34795   86.75067  115.37799  145.80942
##  [7]  177.71969  210.84865  244.98322  279.94581  315.58641  351.77704
## [13]  388.40761  425.38284  462.61985  500.04626  537.59870  575.22150
## [19]  612.86572  650.48828  688.05123  725.52112  762.86851  800.06749
## [25]  837.09530  873.93196  910.56001  946.96423  983.13136 1019.04998
## [31] 1054.71026 1090.10382 1125.22356 1160.06358 1194.61902 1228.88596
```

Below is the plot of the predictions along with some original data right before the prediction starts. The shaded area represents the uncertainty of the predictions.

```
pred.time = (n+1):(n+36)
pred.part = sapply(pred.time, pred)
predupper = pred.part + 2*predsd
predlower = pred.part - 2*predsd
yupper = c(y, predupper)
ylower = c(y, predlower)

start = 651 # we only plot observations after this time (inclusive)
df = data.frame(
  t = start:(n+36),
  y = y.pred[start:(n+36)],
  yupper = yupper[start:(n+36)],
  ylower = ylower[start:(n+36)]
)

ggplot(df) +
  geom_line(aes(x=t, y=y)) +
  geom_vline(xintercept = n, color='red') +
  geom_ribbon(aes(x=t, ymin=ylower, ymax=yupper), alpha=.2)
```



The predictions make sense: because the largest root β is very close to 1, the prediction graph is almost linear, and the slope looks similar to the slope of the data. Also, the uncertainty interval is wider for predictions that are farther in the future.

d

Suppose that we want to predict the US population for the months preceding January 1959. Compare your fitted backward model with the forward model fitted earlier. Are there any similarities between the two models? (4 points)

```
Xmat = matrix(1, nrow=n-2, ncol=3)
Xmat[,2] = y[2:(n-1)]
Xmat[,3] = y[3:n]
modar.back = lm(y[1:(n-2)] ~ -1 + Xmat)
summary(modar.back)
```

```
##
## Call:
## lm(formula = y[1:(n - 2)] ~ -1 + Xmat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -52.782 -12.146   2.159  10.397 179.276
##
## Coefficients:
```

```
##        Estimate Std. Error t value Pr(>|t|)
## Xmat1 -11.58669    4.35820  -2.659  0.00801 **
## Xmat2   1.94820    0.01132 172.030  < 2e-16 ***
## Xmat3  -0.94820    0.01133 -83.715  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.72 on 757 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 6.135e+10 on 3 and 757 DF,  p-value: < 2.2e-16
```

Compare the coefficients of the forward model with the coefficients of the backward model:

```
modar$coefficients
```

```
##      Xmat1      Xmat2      Xmat3
## 12.9892477  1.9518065 -0.9518189
```

```
modar.back$coefficients
```

```
##       Xmat1       Xmat2       Xmat3
## -11.5866876   1.9482016  -0.9481988
```

We see that the intercepts have roughly the same magnitude but have opposite sign, while the other coefficients are quite similar. The reason why the intercepts have opposite sign might be because of going foward versus going backward.

e

Using your model from the previous part, predict the US population for the 36 months immediately preceding January 1959. Plot these predictions and uncertainty indicators along with the original data. Do these predictions make intuitive sense? (6 points).

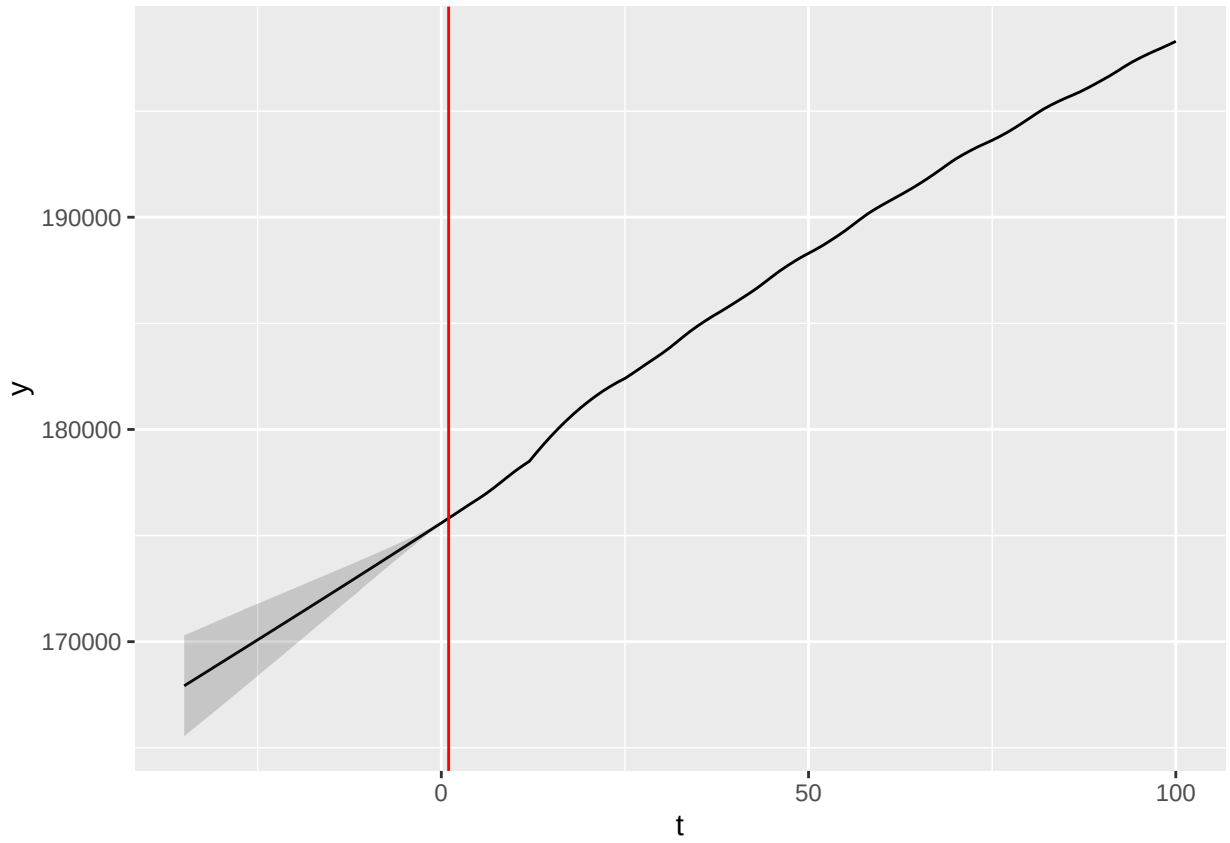
```
y.full = c(rep(-9999,36), y) # y prepended with the 36 backward predictions
k = 36
while(k>0) {
  y.full[k] = sum(modar.back$coefficients * c(1, y.full[k+1], y.full[k+2]))
  k = k-1
}

sighat = sqrt((sum((modar.back$residuals)^2))/(n-2*2-1)) #note: we use modar.back here
Gamhat = matrix(sighat^2, 1, 1)
vkp = matrix(modar.back$coefficients[2], 1, 1) #note: we use modar.back here
for(i in 1:(36-1))
{
    covterm = Gamhat %*% vkp
    varterm = (sighat^2) + (t(vkp) %*% (Gamhat %*% vkp))
    Gamhat = cbind(Gamhat, covterm)
    Gamhat = rbind(Gamhat, c(t(covterm), varterm))

    # update vkp
```

```
    if (i < p) {vkp = c(modar.back$coefficients[(i+2)], vkp)}
    if (i >= p) {vkp = c(0, vkp)}
}
predsd = sqrt(diag(Gamhat))
predsd = rev(predsd) # reverse order because the prediction goes backwards
predsd
```

```
##  [1] 1186.43397 1154.03097 1121.32924 1088.33077 1055.03841 1021.45593
##  [7]  987.58820  953.44128  919.02259  884.34104  849.40724  814.23372
## [13]  778.83513  743.22852  707.43367  671.47342  635.37406  599.16580
## [19]  562.88335  526.56647  490.26079  454.01862  417.90007  381.97427
## [25]  346.32098  311.03250  276.21613  241.99739  208.52416  175.97240
## [31]  144.55419  114.52950   86.22450   60.06199   36.61663   16.72099
```

In the plot below, predictions start on the left side of the red line (not including the red line):

```
yupper = y.full + c(2*predsd, rep(0, n))
ylower = y.full - c(2*predsd, rep(0, n))
end = 100 # we only plot observations before this time
df = data.frame(
  t = (-35):end,
  y = y.full[1:(36+end)],
  yupper = yupper[1:(36+end)],
  ylower = ylower[1:(36+end)]
)

ggplot(df) +
  geom_line(aes(x=t, y=y)) +
  geom_vline(xintercept = 1, color='red') +
  geom_ribbon(aes(x=t, ymin=ylower, ymax=yupper), alpha=.2)
```



The predictions make sense: the prediction graph is almost linear, and the slope looks similar to the slope of the data. Also, the uncertainty interval is wider for predictions that are further away from the data.

3

```
y = pop$POPTHM
n = length(y)
```

a

Fit the AR(3) model to the data.

```
# fits y to AR(p) model
fit.ar = function(y, p) {
  n = length(y)
  Xmat = matrix(1, n-p, p+1)
  for(j in 1:p)
  {
     Xmat[, (j+1)] = y[(p+1-j):(n-j)]
  }
  linmod = lm(y[(p+1):n] ~ -1 + Xmat)
  return( linmod )
}

# I first used arima(y, order=c(3,0,0)), but it seems to give wrong coefficients?
```

The following are point estimates of phi0, phi1, phi2, phi3:

```
modar = fit.ar(y, 3)
modar$coefficients
```

```
##      Xmat1      Xmat2      Xmat3      Xmat4
## 17.9023526  2.3568034 -1.7847529  0.4279367
```

The following is a point estimate for sigma:

```
sqrt((sum((modar$residuals)^2))/(n-3*2-1))
```

```
## [1] 15.18037
```

b

For better interpretability, I want to fit a model to the twice differenced series. Compare this twice-difference model to the AR(3) model fitted in the previous part. Are they similar?

```
diff = rep(0, n) # twice difference of the time series
for (i in 3:n) {
  diff[i] = y[i] - 2*y[i-1] + y[i-2]
}
d = diff[3:n]

modar.d = fit.ar(d, 1)
alpha0 = modar.d$coefficients[1] %>% as.numeric()
alpha1 = modar.d$coefficients[2] %>% as.numeric()
alpha0
```

```
## [1] -0.08859117
```

```
alpha1
```

```
## [1] 0.3898093
```

The following four numbers are the coefficients obtained from rewriting the twice-diffrence model back in terms of Y_t:

```
alpha0
```

```
## [1] -0.08859117
```

```
alpha1 + 2 # close to phi1
```

```
## [1] 2.389809
```

```
-1-2*alpha1 # close to phi2
```

```
## [1] -1.779619
```

```
alpha1 # close to phi3
```

```
## [1] 0.3898093
```

Except that alpha0 is not close to phi0, the other coefficients are similar in both models. It makes sense that alpha0 is not close to phi0 because the twice-difference operation kills any linear trend in the time series.

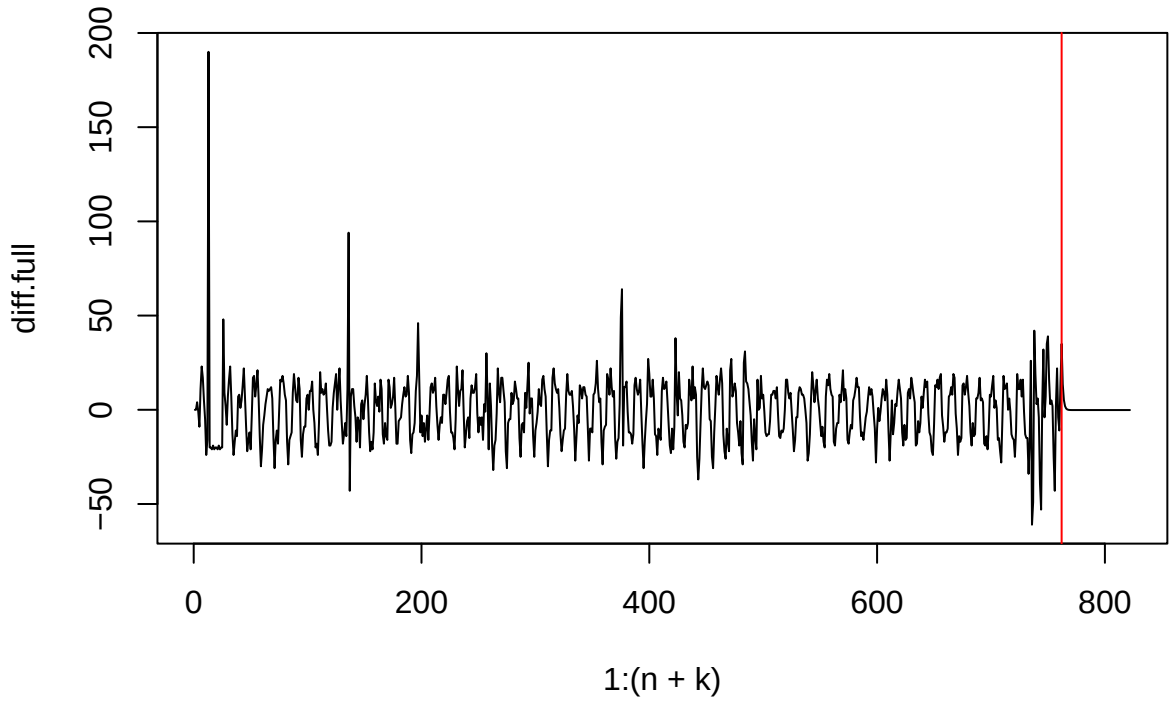
c

Compare the predictions of the two models for the next 60 time points. Are the predictions similar? (4 points).

```r
k = 60

# compute AR(3) predictions
y.full = c(y, rep(-9999, k))
for (t in (n+1):(n+k)) {
  phis = modar$coefficients %>% as.numeric()
  #y.full[t] = sum(as.numeric(ar3$coef) * c(y.full[t-1], y.full[t-2], y.full[t-3], 1))
  y.full[t] = sum(phis * c(1, y.full[t-1], y.full[t-2], y.full[t-3]))
}
```

Below are predictions for the twice-difference D_t:

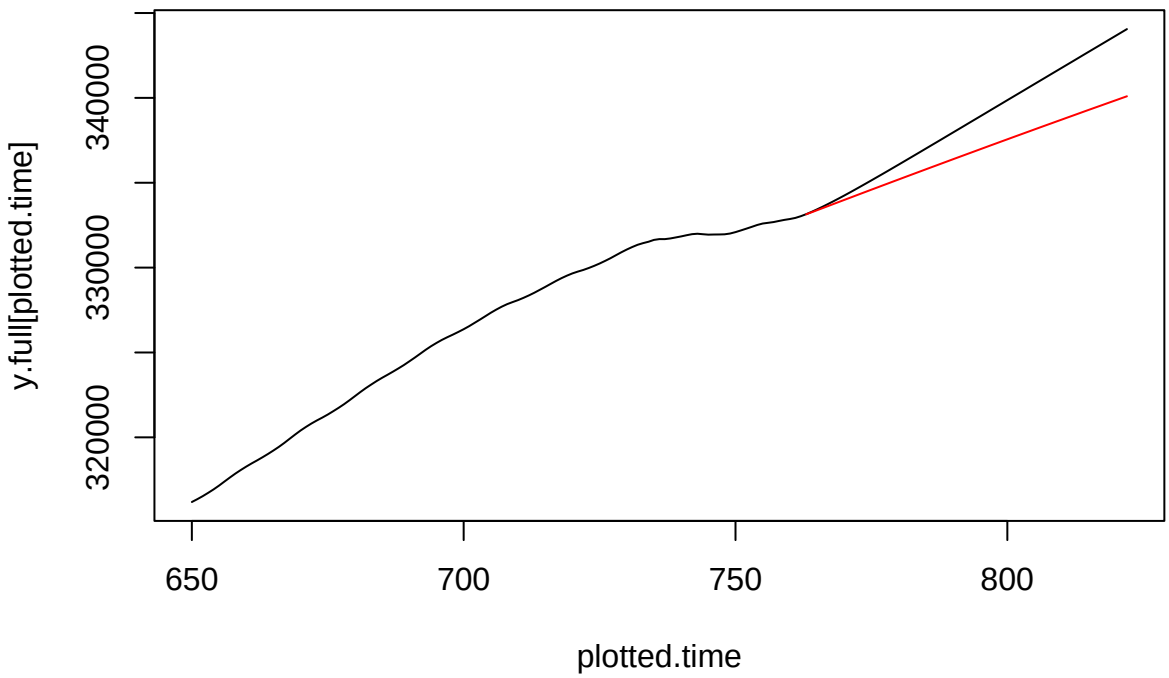
```r
diff.full = c(diff, rep(-9999, k))
for (t in (n+1):(n+k)) {
  alphas = modar.d$coefficients %>% as.numeric()
  diff.full[t] = sum(alphas * c(1, diff.full[t-1]))
}
plot(1:(n+k), diff.full, type='l')
abline(v=n, col='red')
```



Then, Y_t can be recovered from D_t as follows:

```r
y.d = c(y, rep(-9999, k)) # d stands for difference
for (t in (n+1):(n+k)) {
  y.d[t] = diff.full[t] + 2*y.d[t-1] - y.d[t-2]
}
```

```
start = 650
plotted.time = start:(n+k)
pred.time = (n+1):(n+k)
plot(plotted.time, y.full[plotted.time], type='l')
lines(pred.time, y.d[pred.time], col='red')
```



Both predictions are quite linear and increasing. But the predictions given by the twice-difference model are smaller and have a smaller slope. This happens because all the future predictions by the twice-difference model are determined by the last two entries of the data `y` and the twice-difference model stores no information about the linear trend of the data. Hence, the future slope is largely determined by the slope given by the last two entries of the data. To take the linear trend of the whole data into account, one should consider the residuals of `y` obtained from linear regression in the first place, then do the same process as above to get predictions for the residuals, and then add back the linear part of the linear regression.

4

Download the FRED dataset on "Retail Sales: Beer, Wine, and Liquor Stores" from https://fred.stlouisfed.org/series/MRTSSM4453USN. This is a monthly dataset (the units are millions of dollars) and is not seasonally adjusted. Separate the last 48 observations from this dataset and keep them as a test dataset. Fit the AR(p) model for each p = 1, 2, . . . , 24 for the training dataset and use it to predict the observations in the test dataset. Evaluate the 24 models based on the accuracy of prediction and report the model with the best prediction accuracy. (10 points).

```
alc = read.csv('alc.csv')
colnames(alc)[2] = 'y'
```

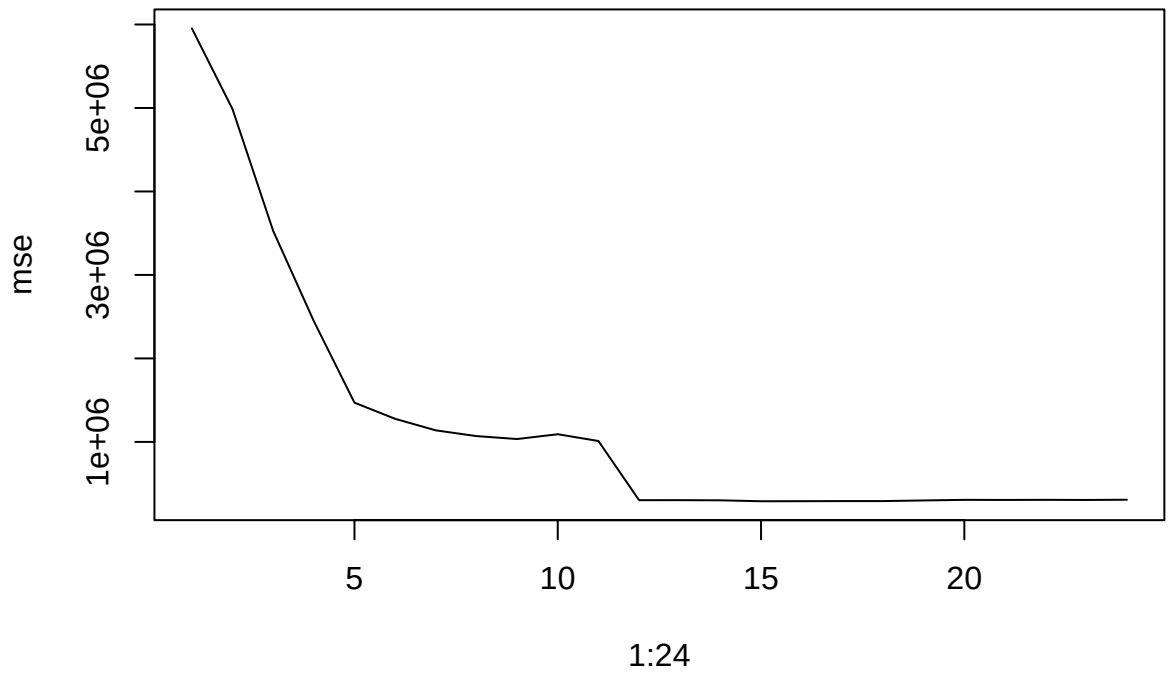
```
n.test = 48
n.train = nrow(alc) - n.test
y.train = alc$y[1:n.train]
y.test = alc$y[(n.train+1):nrow(alc)]
```

Let's choose p based on the mean square error of the predictions:

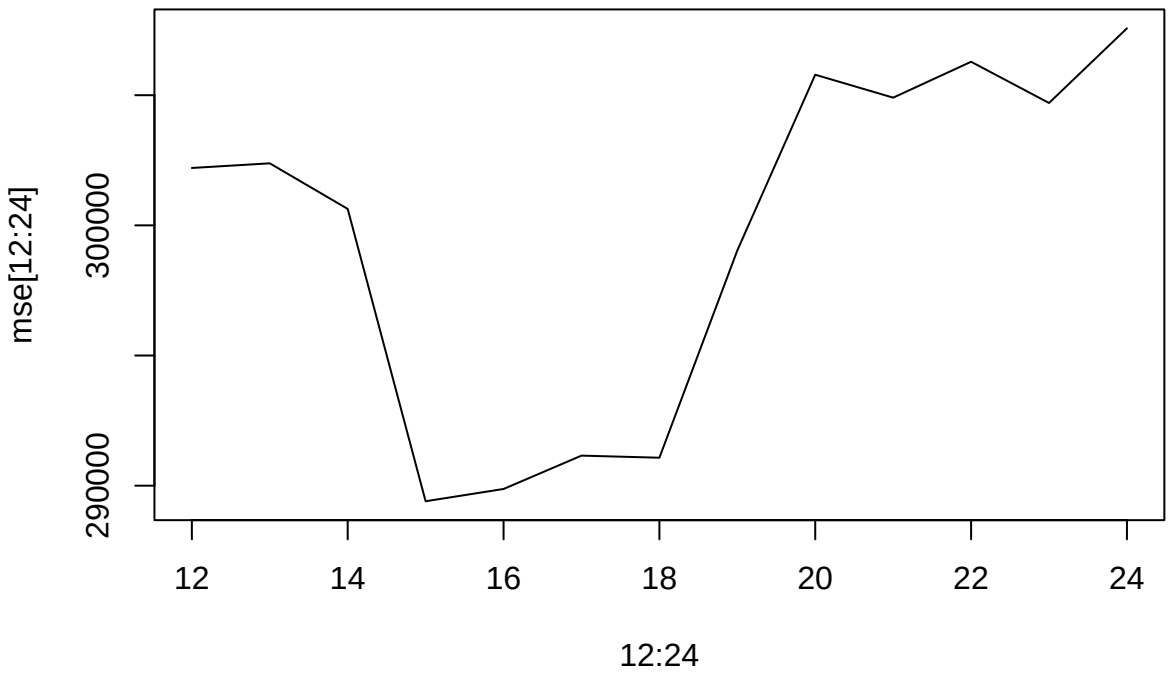
```
mse = rep(-1, 24)
for (p in 1:24) {
  Xmat = matrix(1, n.train-p, p+1)
  for(j in 1:p) {
     Xmat[, (j+1)] = y.train[(p+1-j):(n.train-j)]
  }
  linmod = lm(y.train[(p+1):n.train] ~ -1 + Xmat)

  y.pred = c(y.train, rep(-9999, n.test))
  for (t in (n.train+1):nrow(alc)) {
    y.pred[t] = sum(c(1, y.pred[(t-1):(t-p)]) * linmod$coefficients)
  }

  y.pred = y.pred[(n.train+1):nrow(alc)]
  mse[p] = mean((y.pred - y.test)^2)
}
```

The first plot reveals a significant drop of MSE when p=12. The second plot shows that p=15 attains the lowest MSE, so we will choose p=15.

```
plot(1:24, mse, type='l')
```

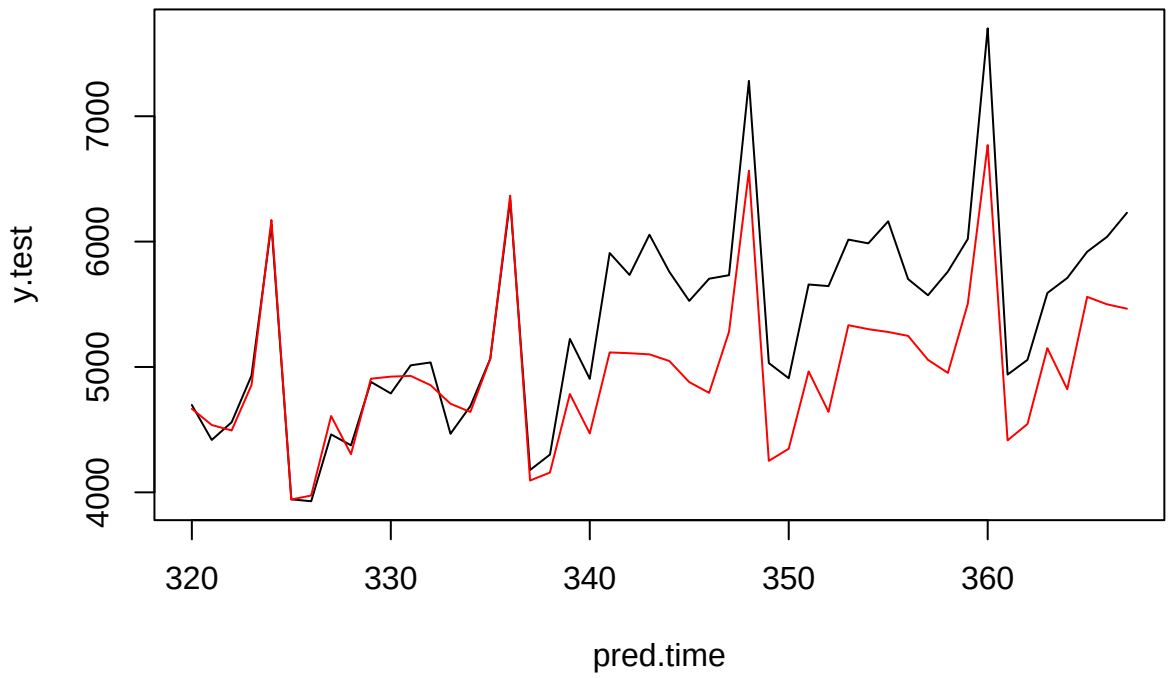



```
plot(12:24, mse[12:24], type='l')
```



Plot predictions vs. actual data to compare, where predictions are colored red:

```
p = 15
Xmat = matrix(1, n.train-p, p+1)
for(j in 1:p)
{
   Xmat[, (j+1)] = y.train[(p+1-j):(n.train-j)]
}
linmod = lm(y.train[(p+1):n.train] ~ -1 + Xmat)

y.pred = c(y.train, rep(-9999, n.test))
for (t in (n.train+1):nrow(alc)) {
  y.pred[t] = sum(c(1, y.pred[(t-1):(t-p)]) * linmod$coefficients)
}
pred.time = (n.train+1):nrow(alc)
y.pred = y.pred[pred.time]
plot(pred.time, y.test, type='l') # actual data
lines(pred.time, y.pred, col='red')
```


y.test
4000
5000
6000
7000
320
330
340
350
360
pred.time