Homework 6 Solutions

Due: Tuesday 3/10/20 by 10:00am

This week, we'll try modeling dependence over time using an moving average model of order q:

$$y_t = \mu + \sum_{i=1}^{q} \theta_i w_{t-i} + w_t, \quad w_t \overset{i.i.d.}{\sim} \text{normal}\left(0, \sigma_w^2\right) \tag{1}$$

Assume y_t is a stationary process, i.e. assume that the process has constant mean $\mathbb{E}\left[y_t\right] = \mu$, finite variance $\mathbb{V}\left[y_t\right] = \gamma_y\left(0\right)$, and autocovariances $\text{Cov}\left[y_t, y_{t-h}\right] = \gamma_y\left(\left|h\right|\right)$.

1. Checking Invertability and Examining the Autocorrelation Function

(a) First, let's explore how to check if values of $\theta_1, \dots, \theta_q$ correspond to an invertible process.

The `R` library `polynom` lets us easily compute the roots of polynomials. You'll need to install the `polynom` library and load it. Here's a little example:

```
library(polynom)

# Create a "polynomial" object for the polynomial
# 1 - 5x + 3x^2 + 2x^3
pol <- polynomial(c(1, -5, 3, 2))
# Get the values of x for which 1 - 5x + 3x^2 + 2x^3 = 0
sol <- solve(pol)
```

You may get complex roots $r = a + bi$. Note that the absolute value of a complex number r is given by $\left|r\right| = \sqrt{a^2 + b^2}$.

Consider the following $\mathbf{MA}(q)$ models, all with $\sigma_w^2 = 1$.

i. $q = 1$, $\theta_1 = 8.11$
ii. $q = 2$, $\theta_1 = -5.64$, $\theta_2 = -9.34$
iii. $q = 2$, $\theta_1 = 2.15$, $\theta_2 = 3.69$
iv. $q = 3$, $\theta_1 = 2.08$, $\theta_2 = 3.39$, $\theta_3 = 0.73$

(a) For (i)-(iii), find the root of the moving average polynomial that is smallest in magnitude by solving $\theta\left(z\right) = 0$ for z by hand, without using any special `R` functions. For (iv), use `polynom` to find the root that is smallest in magnitude. Give the value of this root and indicate whether or not the model is invertible.

The smallest roots in absolute value for each MA polynomial are 0.12, 0.14, 0.52, and 0.58, respectively. Because the absolute values of these roots are inside of the unit circle, none of the four models are invertible.

```
pars <- list(c(8.11), c(5.64, -9.34), c(2.15, 3.69), c(2.08, 3.39, 0.73))
for (i in 1:length(pars)) {
  theta.z <- polynomial(c(1, pars[[i]]))
  sol <- solve(theta.z)
}
```

(b) Compute the autocorrelation function for (i)-(iv) for values of $h \geq 0$ by hand.

Let's work out the autocorrelation function for a general MA(3) process,

$$y_t = \mu + \theta_1 w_{t-1} + \theta_2 w_{t-2} + \theta_3 w_{t-3} + w_t.$$

First we need to work out the autocovariance function. We know that only $\gamma(0)$, $\gamma(1)$, $\gamma(2)$, and $\gamma(3)$ will be nonzero.

$$\begin{aligned}
\gamma(0) &= \left(1 + \theta_1^2 + \theta_2^2 + \theta_3^2\right)\sigma_w^2 \\
\gamma(1) &= \left(\theta_1 + \theta_1\theta_2 + \theta_2\theta_3\right)\sigma_w^2 \\
\gamma(2) &= \left(\theta_2 + \theta_1\theta_3\right)\sigma_w^2 \\
\gamma(3) &= \theta_3\sigma_w^2
\end{aligned}$$

It follows that the autocorrelation function $\rho(h)$ is equal to 0 when $|h| > 3$ and equal to the following otherwise:

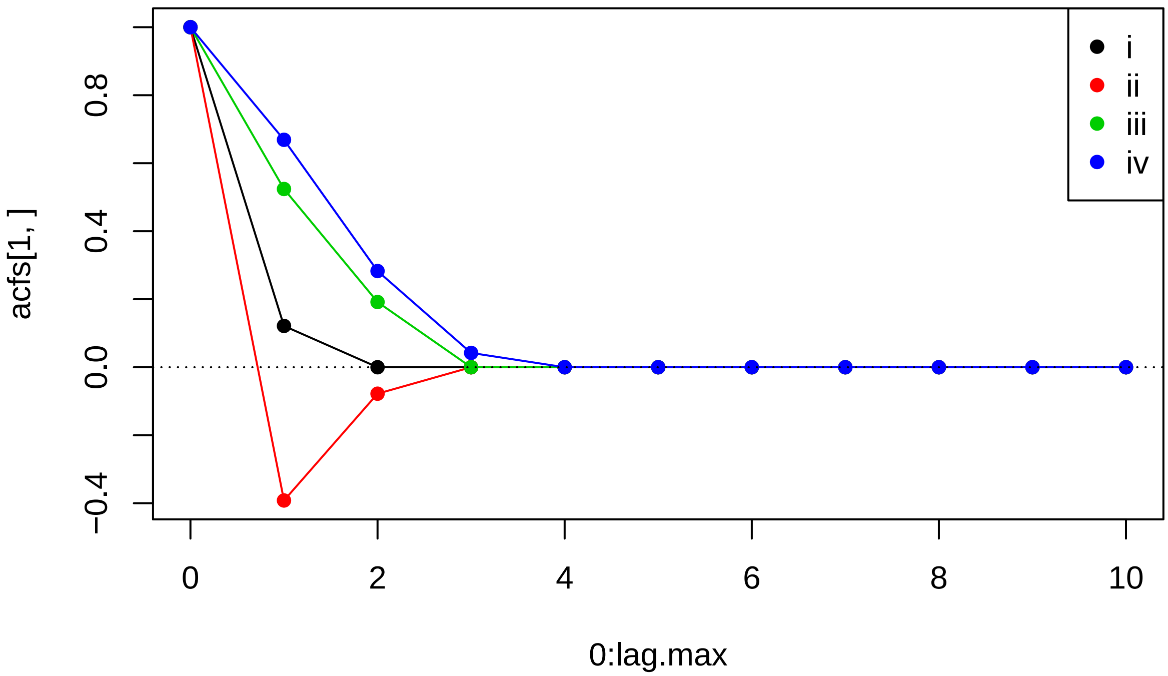
$$\begin{aligned}
\rho(0) &= 1 \\
\rho(1) &= \frac{\theta_1 + \theta_1\theta_2 + \theta_2\theta_3}{1 + \theta_1^2 + \theta_2^2 + \theta_3^2} \\
\rho(2) &= \frac{\theta_2 + \theta_1\theta_3}{1 + \theta_1^2 + \theta_2^2 + \theta_3^2} \\
\rho(3) &= \frac{\theta_3}{1 + \theta_1^2 + \theta_2^2 + \theta_3^2}
\end{aligned}$$

We can plug in specific values for θ_1, θ_2, and θ_3 to get the autocorrelation function for each model.

i. $\rho(0) = 1$, $\rho(1) = 0.12$, $\rho(h) = 0$ for $h > 1$.
ii. $\rho(0) = 1$, $\rho(1) = -0.39$, $\rho(2) = -0.08$, $\rho(h) = 0$ for $h > 2$.
iii. $\rho(0) = 1$, $\rho(1) = 0.52$, $\rho(2) = 0.19$, $\rho(h) = 0$ for $h > 2$.
iv. $\rho(0) = 1$, $\rho(1) = 0.67$, $\rho(2) = 0.28$, $\rho(3) = 0.04$, $\rho(h) = 0$ for $h > 3$.

(c) Using `ARMAacf` to compute $\rho_x(h)$, plot the autocorrelation function $\rho_x(h)$ for $h = 0, \dots, 10$ for the **MA**(q) models on a single plot. Include a dotted horizontal line at 0.

```
lag.max <- 10
acfs <- matrix(nrow = length(pars), ncol = lag.max + 1)
for (i in 1:length(pars)) {
  acfs[i, ] <- ARMAacf(ma = pars[[i]], lag.max = lag.max)
}
plot(0:lag.max, acfs[1, ], type = "n", ylim = range(acfs))
for (i in 1:length(pars)) {
  points(0:lag.max, acfs[i, ], col = i, pch = 16)
  lines(0:lag.max, acfs[i, ], col = i)
}
abline(h = 0, lty = 3)
legend("topright", col = 1:4, pch = rep(16, 4), legend = c("i", "ii", "iii", "iv"))
```



(d) In general, it isn't straightforward to take an non-invertible MA(q) model and transform it to an invertible MA(q) model. By straightforward, I mean that there is not simple function that takes non-invertible MA(q) parameters and returns the corresponding invertible MA(q) that yield the same autocorrelation function. Fortunately, we can use `R` to do this numerically by taking a set of possibly non-invertible moving average parameters $\theta_1, \dots, \theta_q$ and:

- Using `ARMAacf` to compute the autocorrelation function $\rho_y(h)$ for $h = 0, \dots, h_{max}$ for some h_{max};
- Using `acf2AR` to find the stationary AR(h_{max}) parameters that perfectly characterize the autocorrelation function for $\rho_y(h)$ for $h = 0, \dots, h_{max}$ for some h_{max};
- Using `ARMAtoMA` to take the AR(h_{max}) parameters and compute the corresponding *invertible* moving average MA(q) model (remember - every autoregressive model has a possibly infinite order moving average representation).

To check your work, you can compare the autocorrelation functions $\rho_y(h)$ for $h = 0, \dots, h_{max}$ obtained using `ARMAacf` for the possibly non-invertible MA(q) model, the AR(h_{max}) representations, and the invertible MA(q) model.

Provide the invertible representations of the non-invertible MA(q) models in (a).

```
pars <- list(c(8.11),
             c(5.64, -9.34),
             c(2.15, 3.69),
             c(2.08, 3.39, 0.73))
lag.max = 100
for (i in 1:length(pars)) {
  true.acf <- ARMAacf(ma = pars[[i]], lag.max = lag.max)
  ar.acf <- acf2AR(true.acf)[lag.max, ]
  ma.coef <- ARMAtoMA(ar = ar.acf,
                      lag.max = lag.max)
}
```

i. $q = 1$, $\theta_1 = 0.12$
ii. $q = 2$, $\theta_1 = -0.60$, $\theta_2 = -0.11$
iii. $q = 2$, $\theta_1 = 0.58$, $\theta_2 = 0.27$
iv. $q = 3$, $\theta_1 = 0.87$, $\theta_2 = 0.50$, $\theta_3 = 0.08$

2. We're going to keep working with residuals from the `broc` data for a little while longer. Again, it is posted on the course website, which contains the average price of one pound of broccoli in urban areas each month, from July 1995 through December 2019. Throughout this problem, we'll continue to work with the residuals from fitting a linear model with a linear time trend and month effects to all but the last 12 months of data. We'll call them `y`, because we'll be thinking of them as our observed time series, and we'll define $m = n - 12$.

```
load("~/Dropbox/Teaching/TimeSeries2020/stat697/content/data/broc.RData")
set.seed(1)
broc$fdate <- as.Date(broc$date, "%Y-%m-%d")
broc$month <- format(broc$fdate, "%m")
broc$dayssincestart <- as.numeric(broc$fdate) - min(as.numeric(broc$fdate))
n <- nrow(broc)
m <- nrow(broc) - 12
linmod <- lm(price~dayssincestart+factor(month), data = broc,
             subset = 1:m)
pred <- predict(linmod, broc)
y <- broc$price - pred
```

(a) Now let's fit some MA(q) models using `arima` again. We are going to consider the same models that we considered on the previous homework. For $q = 1, 2, 4, 8, 16, 32$, compute estimates of μ, $\theta_1, \dots, \theta_q$ using `arima`. Make a plot with 6 panels. Using one panel for each value of q, plot the last 24 observations and the estimated fitted values $\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]$ from the corresponding fitted model obtained using the Durbin-Levinson algoirthm, and the approximate 95% confidence intervals for the estimated fitted values obtained using the parametric bootstrap. You can use `arima.sim` to simulate MA(q) processes with the mean, moving average parameters, and variance that you estimated from the data. I recommend you construct your 95% intervals by using the parametric bootstrap to approximate $\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]|y_1, \dots, y_{t-1}\right]$ and then assuming that $\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]$ is approximately normal - i.e. a $1 - \alpha$ interval computed in this way would be:

$$\left(\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right] + z_{\alpha/2}\sqrt{\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]|y_1, \dots, y_{t-1}\right]}, \right.$$
$$\left. \widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right] + z_{1-\alpha/2}\sqrt{\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]|y_1, \dots, y_{t-1}\right]}\right).$$

Code for implementing the Durbin-Levinson algorithm is given below.

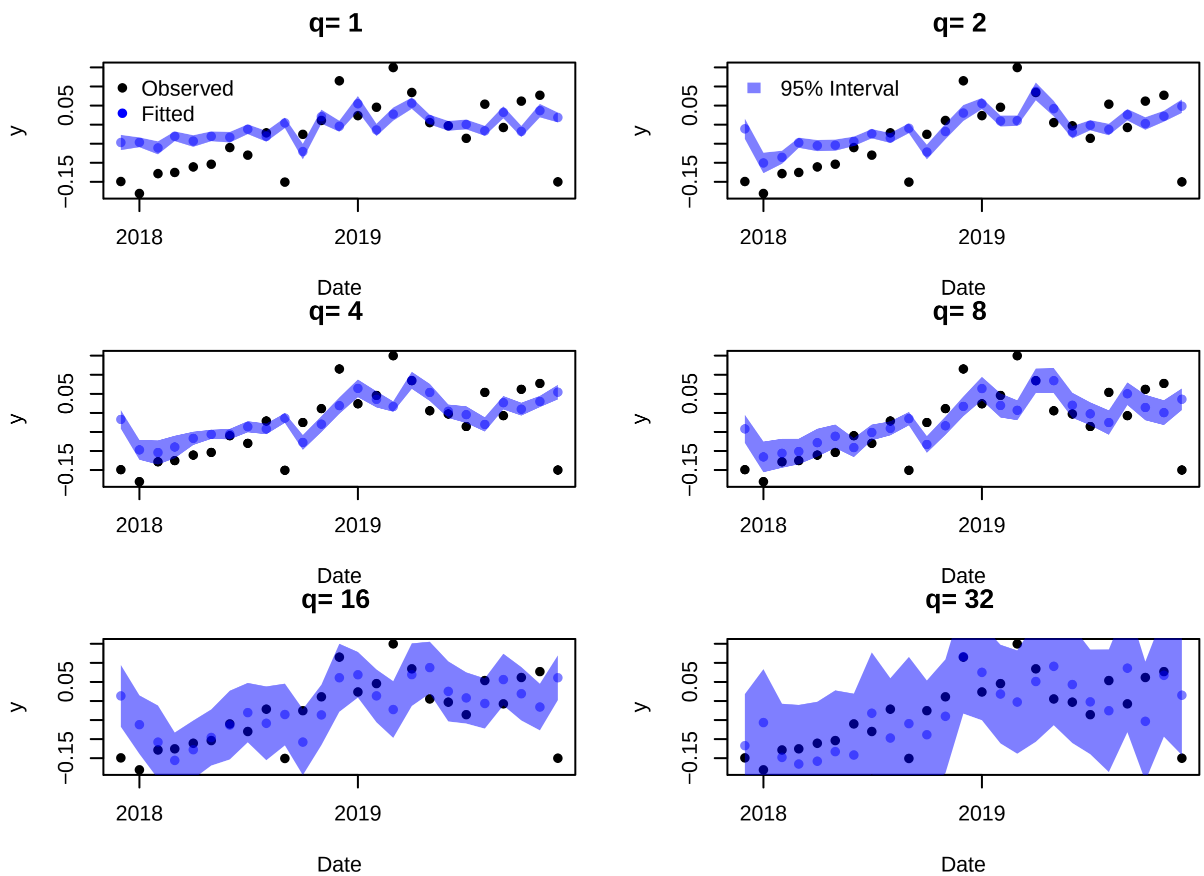
```
solve.dl <- function(n, theta = 0, phi = 0, sig.sq.w = 1) {
  C <- matrix(nrow = n, ncol = n)
  v <- rep(NA, n + 1)
  marep <- ARMAtoMA(ar = phi, ma = theta, lag.max = 1000)
  gamma.x <- ARMAacf(ar = phi, ma = theta,
                     lag.max = n)*(1 + sum(marep^2))*sig.sq.w
  gamma.x.0 <- gamma.x[1]
  C[1, 1] <- gamma.x[2]/gamma.x.0
  v[1] <- gamma.x.0
  v[2] <- v[1]*(1 - C[1, 1]^2)
  for (i in 2:n) {
    C[i, i] <- gamma.x[i + 1]
    for (j in 1:(i - 1)) {
      C[i, i] <- C[i, i] - C[i-1, j]*gamma.x[i - j + 1]
    }
    C[i, i] <- C[i, i]/v[i]
    for (j in (i-1):1) {
      C[i, j] <- C[i - 1, j] - C[i, i]*C[i - 1, i - j]
```

```
    }
    v[i + 1] <- v[i]*(1 - C[i, i]^2)
  }
  return(list("C"=C, "c.n" = C[nrow(C), ],
              "v" = v, "v.n" = v[length(v)]))
}
```

```
qs <- 2^(0:5)
ses <- fits <- array(NA, dim = c(length(y), length(qs)))
nboot <- 100

for (q in qs) {
  mamod <- arima(y[1:m], order = c(0, 0, q))
  mu.hat <- mamod$coef[q + 1]
  theta.hat <- mamod$coef[1:q]
  sig.sq.w.hat <- mamod$sigma2
  dl.coef <- solve.dl(n = n, theta = theta.hat, sig.sq.w = sig.sq.w.hat)
  for (i in 2:n) {
      fits[i, which(q == qs)] <- mu.hat +
        sum(dl.coef$C[i, 1:(i - 1)]*(y[(i - 1):1] - mu.hat))
  }

  fits.boot <- matrix(NA, nrow = n, ncol = nboot)
  for (j in 1:nboot) {
    y.sim <- arima.sim(model = list(ma = theta.hat),
                       n = n,
                       sd = sqrt(sig.sq.w.hat)) + mu.hat
    mamod.sim <- try(arima(y.sim[1:m], order = c(0, 0, q)), silent = TRUE)
    if (!class(mamod.sim) == "try-error") {
    mu.hat.sim <- mamod.sim$coef[q + 1]
    theta.hat.sim <- mamod.sim$coef[1:q]
    sig.sq.w.hat.sim <- mamod.sim$sigma2
    dl.coef.sim <- solve.dl(n = n, theta = theta.hat.sim, sig.sq.w = sig.sq.w.hat.sim)
  for (i in 2:n) {
      fits.boot[i, j] <- mu.hat.sim +
        sum(dl.coef.sim$C[i, 1:(i - 1)]*(y[(i - 1):1] - mu.hat.sim))
  }
    }
  }
  ses[, which(q == qs)] <- apply(fits.boot, 1, sd, na.rm = TRUE)
}
```

```
par(mfrow = c(3, 2))
par(mar = c(4, 4, 2, 2))
for (q in qs) {
plot(broc$fdate[(length(y) - 24):length(y)],
     y[(length(y) - 24):length(y)], pch = 16,
     xlab = "Date", ylab = "y",
     main = paste("q=", q, "\n"))
points(broc$fdate[(length(y) - 24):length(y)],
       fits[, which(q == qs)][(length(y) - 24):length(y)],
       col = rgb(0, 0, 1, 0.5), pch = 16)
polygon(c(broc$fdate[(length(y) - 24):length(y)],
          rev(broc$fdate[(length(y) - 24):length(y)])),
```

```
        c(fits[, which(q == qs)][(length(y) - 24):length(y)] +
            qnorm(0.025)*ses[, which(q == qs)][(length(y) - 24):length(y)],
          rev(fits[, which(q == qs)][(length(y) - 24):length(y)] +
                 qnorm(0.975)*ses[, which(q == qs)][(length(y) - 24):length(y)])),
        border = FALSE, col = rgb(0, 0, 1, 0.5))
if (q == min(qs)) {
  legend("topleft", pch = c(16, 16),
         col = c("black", "blue"),
         legend = c("Observed", "Fitted"),
         bty = "n")
}
if (q == qs[2]) {
  legend("topleft",
         fill = rgb(0, 0, 1, 0.5),
         legend = c("95% Interval"),
         bty = "n", border = "white")
}
}
```



(b) Now for comparison, let's fit some AR(p) models using `arima` again again. We are going to consider the same models that we considered on the previous homework. For $p = 1, 2, 4, 8, 16, 32$, compute estimates of μ, $\phi_1, \dots, \phi_p$ using `arima`. Make a plot with 6 panels. Using one panel for each value of p, plot the last 24 observations and the estimated fitted values $\widehat{\mathbb{E}}\left[y_t|y_1, \dots, y_{t-1}\right]$ from the corresponding fitted model obtained using the Durbin-Levinson algoirthm, and the approximate 95% confidence intervals for the estimated fitted values obtained using the parametric bootstrap. You can use `arima.sim` to simulate AR(p) processes with the mean, moving average parameters, and variance that you estimated from the data. I recommend you construct your 95% intervals by using the parametric bootstrap to approximate $\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right]|y_1,\dots,y_{t-1}\right]$ and then assuming that $\widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right]$ is approximately normal - i.e. a $1-\alpha$ interval computed in this way would be:

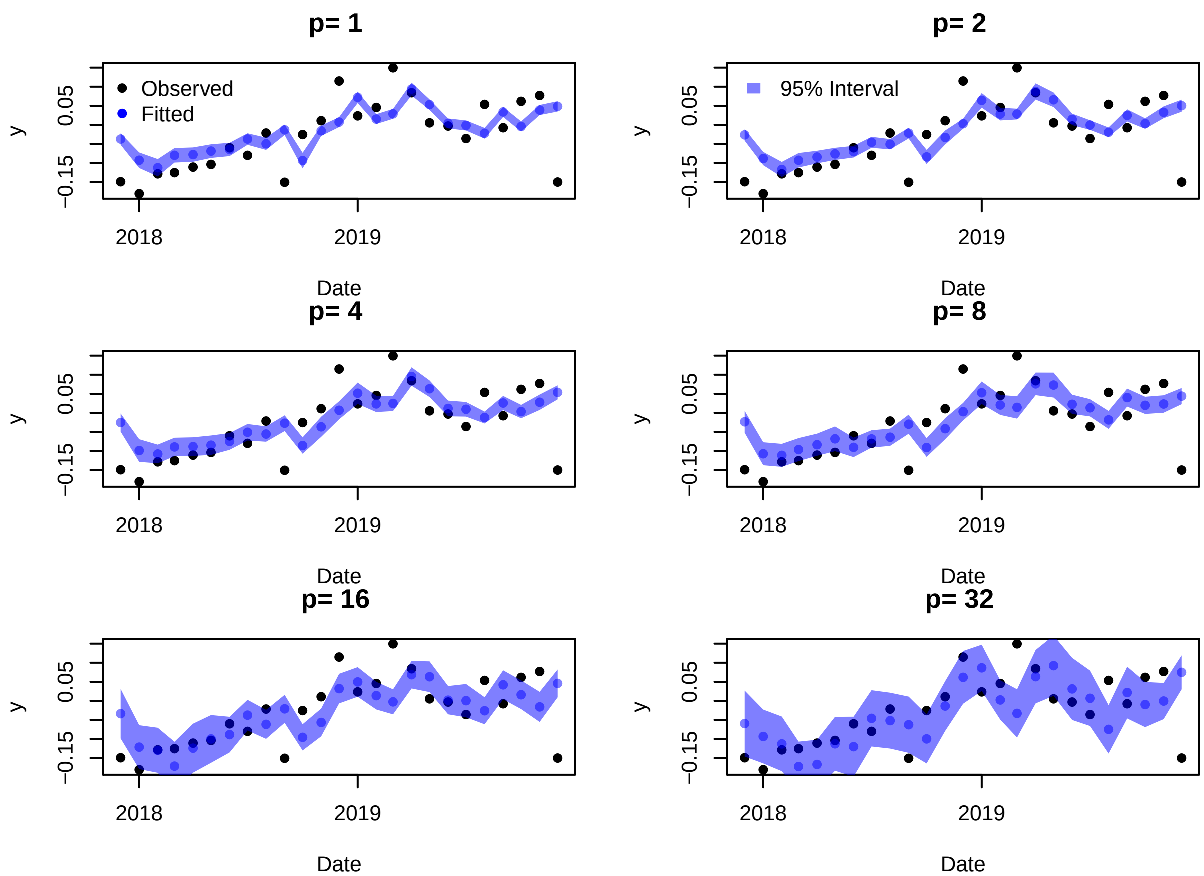
$$\left(\widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right] + z_{\alpha/2}\sqrt{\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right]|y_1,\dots,y_{t-1}\right]}, \right.$$
$$\left. \widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right] + z_{1-\alpha/2}\sqrt{\mathbb{V}\left[\widehat{\mathbb{E}}\left[y_t|y_1,\dots,y_{t-1}\right]|y_1,\dots,y_{t-1}\right]} \right).$$

```
ps <- 2^(0:5)
ses <- fits <- array(NA, dim = c(length(y), length(ps)))
fits <- array(NA, dim = c(n, length(ps)))
nboot <- 100

for (p in ps) {
  armod <- arima(y[1:m], order = c(p, 0, 0))
  mu.hat <- armod$coef[p + 1]
  phi.hat <- armod$coef[1:p]
  sig.sq.w.hat <- armod$sigma2
  dl.coef <- solve.dl(n = n, phi = phi.hat, sig.sq.w = sig.sq.w.hat)
  for (i in 2:n) {
      fits[i, which(p == ps)] <- mu.hat +
        sum(dl.coef$C[i, 1:(i - 1)]*(y[(i - 1):1] - mu.hat))
  }

  fits.boot <- matrix(NA, nrow = n, ncol = nboot)
  for (j in 1:nboot) {
    y.sim <- arima.sim(model = list(ar = phi.hat),
                       n = n,
                       sd = sqrt(sig.sq.w.hat)) + mu.hat
    armod.sim <- try(arima(y.sim[1:m], order = c(p, 0, 0)), silent = TRUE)
    if (!class(armod.sim) == "try-error") {
    mu.hat.sim <- armod.sim$coef[p + 1]
    phi.hat.sim <- armod.sim$coef[1:p]
    sig.sq.w.hat.sim <- armod.sim$sigma2
    dl.coef.sim <- solve.dl(n = n, phi = phi.hat.sim, sig.sq.w = sig.sq.w.hat.sim)
  for (i in 2:n) {
      fits.boot[i, j] <- mu.hat.sim +
        sum(dl.coef.sim$C[i, 1:(i - 1)]*(y[(i - 1):1] - mu.hat.sim))
  }
    }
  }
  ses[, which(p == ps)] <- apply(fits.boot, 1, sd, na.rm = TRUE)
}
```

```
## Warning in log(s2): NaNs produced
```

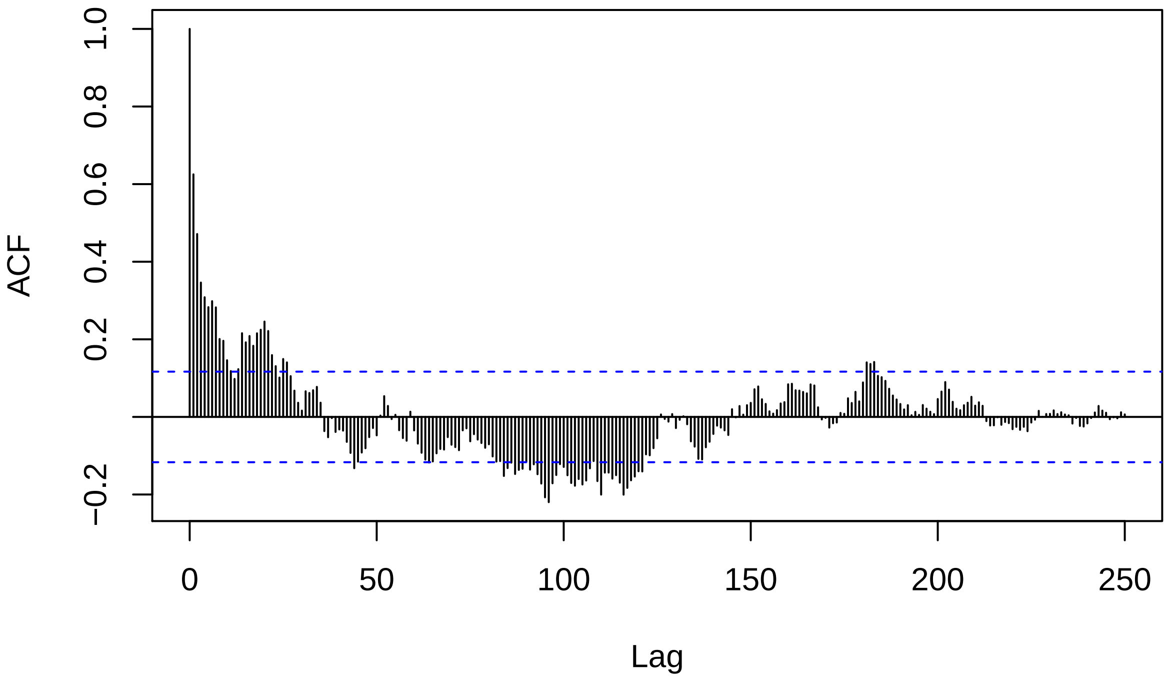
```
par(mfrow = c(3, 2))
par(mar = c(4, 4, 2, 2))
for (p in ps) {
plot(broc$fdate[(length(y) - 24):length(y)],
     y[(length(y) - 24):length(y)], pch = 16,
     xlab = "Date", ylab = "y",
     main = paste("p=", p, "\n"))
points(broc$fdate[(length(y) - 24):length(y)],
```

```r
      fits[, which(p == ps)][(length(y) - 24):length(y)],
      col = rgb(0, 0, 1, 0.5), pch = 16)
polygon(c(broc$fdate[(length(y) - 24):length(y)],
          rev(broc$fdate[(length(y) - 24):length(y)])),
        c(fits[, which(p == ps)][(length(y) - 24):length(y)] +
            qnorm(0.025)*ses[, which(p == ps)][(length(y) - 24):length(y)],
          rev(fits[, which(p == ps)][(length(y) - 24):length(y)] +
                qnorm(0.975)*ses[, which(p == ps)][(length(y) - 24):length(y)])),
        border = FALSE, col = rgb(0, 0, 1, 0.5))
if (p == min(ps)) {
  legend("topleft", pch = c(16, 16),
         col = c("black", "blue"),
         legend = c("Observed", "Fitted"),
         bty = "n")
}
if (p == ps[2]) {
  legend("topleft",
         fill = rgb(0, 0, 1, 0.5),
         legend = c("95% Interval"),
         bty = "n", border = "white")
}
}
```



(c) Plot the autocorrelation function of the first m observations. If you were considering fitting an MA(q) model, what would you choose q to be based on the autocorrelation function alone? Make sure that you choose the maximum value of h for plotting your autocorrelation function $\rho_y(h)$ to reflect your choice.

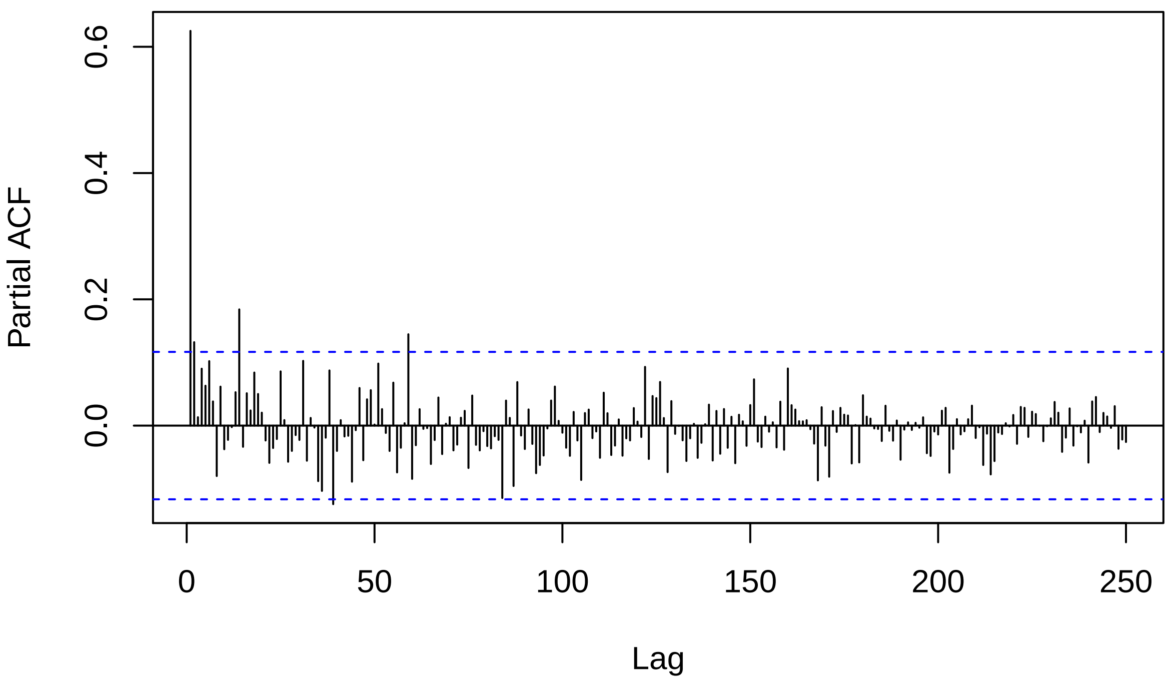
```r
acf(y[1:m], lag.max = 250,
    main = "")
```



Based on the autocorrelation function alone, I would choose $q = 184$.

(d) Plot the partial autocorrelation function of the first m observations. If you were considering fitting an AR(p) model, what would you choose p to be based on the partial autocorrelation function alone? Make sure that you choose the maximum value of h for plotting your autocorrelation function $\rho_y(h)$ to reflect your choice.

```r
pacf(y[1:m], lag.max = 250,
     main = "")
```



Based on the partial autocorrelation function alone, I would choose $p = 59$.

(e) Based on (a), (b), (c), (d), and the results of previous analyses on previous assignments, do you think an AR model or an MA model is more appropriate for this data?

Based on (a) and (b), it appears that all of the autoregressive models perform similarly regardless of the order p, whereas the performance of the moving average models changes more as q increases. This suggests that a relatively small autoregressive model might suffice, whereas a larger moving average model would be needed. We draw similar conclusions from (c) and (d). Based on the autocorrelation and partial autocorrelation functions, we would choose a prohibitively large MA model and a much smaller autoregressive model. Altogether, this suggests that an autoregressive model is likely more appropriate than a moving average model for this data.