Homework 5 Solutions

Due: Thursday 2/27/20 by 10:00am

In last week's homework, tried modeling dependence over time, using an autoregressive model of order p:

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

Note that we did not assume y_t is a stationary process in the previous homework. Now, assume y_t is a stationary process, i.e. assume that the process has constant mean $\mathbb{E}[y_t] = \mu$, finite variance $\mathbb{V}[y_t] = \gamma_y(0)$, and autocovariances $\text{Cov}[y_t, y_{t-h}]\, \gamma_y(|h|)$.

1. Checking Stationarity and Examining the Autocorrelation Function

(a) First, let's explore how to check if values of $\phi_1, \ldots, \phi_p$ correspond to a stationary 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 $|r| = \sqrt{a^2 + b^2}$.

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

i. $p = 1$, $\phi_1 = -0.5$
ii. $p = 2$, $\phi_1 = 0.1$, $\phi_2 = 0.8$
iii. $p = 2$, $\phi_1 = 0.3$, $\phi_2 = -0.9$
iv. $p = 3$, $\phi_1 = 0.8$, $\phi_2 = 0.1$, $\phi_3 = 0.01$

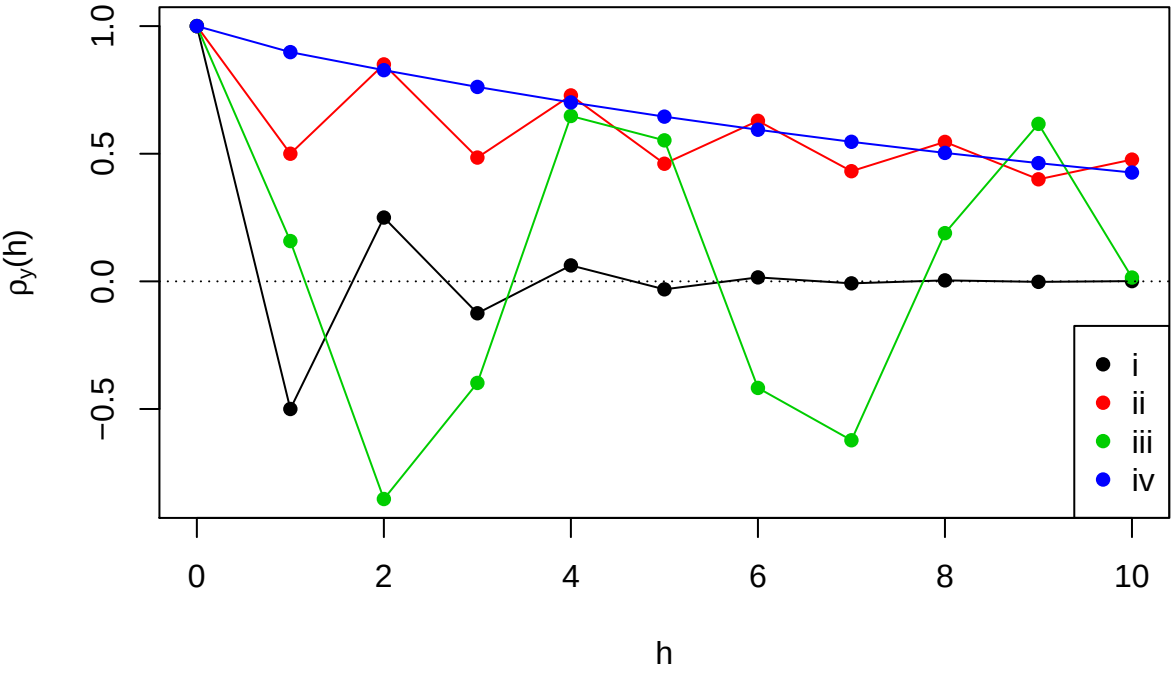
(a) For (i)-(iii), find the root of the autoregressive polynomial that is smallest in magnitude by solving $\phi(z) = 0$ for z by hand, without using any special `R` functions. For (iv), use `polynom` to find the root that is smallest in magnitude. Give the value of this root and indicate whether or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

The smallest roots in absolute value for each AR polynomial are 2.00, 1.06, 1.05, and 1.09, respectively. Because the absolute values of these roots are outside of the unit circle, all four models are stationary.

```
pars <- list(c(-0.5), c(0.1, 0.8), c(0.3, -0.9), c(0.8, 0.1, 0.01))
for (i in 1:length(pars)) {
  phi.z <- polynomial(c(1, -pars[[i]]))
  sol <- solve(phi.z)
}
```

(b) Using `ARMAacf` to compute $\rho_x(h)$, plot the autocorrelation function $\rho_x(h)$ for $h = 0, \ldots, 10$ for the stationary $\mathbf{AR}(p)$ 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(ar = pars[[i]], lag.max = lag.max)
}
plot(0:lag.max, acfs[1, ], type = "n", ylim = range(acfs),
     xlab = "h",
     ylab = expression(paste(rho[y], "(h)", sep = "")))
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("bottomright", col = 1:4, pch = rep(16, 4), legend = c("i", "ii", "iii", "iv"))
```



(c) Based on the plot you made in (b), describe what kinds of values of the autoregressive coefficients $\phi_1, \ldots, \phi_p$ produce the following behaviors:

- $\rho_y(h)$ oscillates between positive and negative values;
- $\rho_y(h)$ oscillates between larger and smaller but always positive values;
- $\rho_y(h)$ decays very quickly in magnitude;
- $\rho_y(h)$ decays very slowly.

Negative values of $\phi_1, \ldots, \phi_p$ appear to produce autocorrelations $\rho_y(h)$ that oscillate between positive and negative values. Values of $\phi_1, \ldots, \phi_p$ close to -1 or 1 appear to produce autocorrelations $\rho_y(h)$ that decay very slowly, whereas values of $\phi_1, \ldots, \phi_p$ close to zero appear to produce autocorrelations $\rho_y(h)$ that decay very quickly in magnitude. Positive values of $\phi_1, \ldots, \phi_p$ that increase with p appear to produce autocorrelations $\rho_y(h)$ that oscillate between larger and smaller but always positive values.

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.

```
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) First, set $p = 1$. Let's define $m = n - 12$ for convenience. Write down the likelihood for $y_1, \ldots, y_m$ as a function of the scalar mean μ, and the corresponding $m \times m$ covariance matrix $\mathbf{\Sigma}$. Compute the log-likelihood for $\sigma_w^2 = 1$, $\phi = 0.25$, and $\mu = 0$ using this way of writing down the log-likelihood, i.e. first construct the autocovariances $\gamma_y(0), \ldots, \gamma_y(m-1)$ from ϕ_1 and σ_w^2, then construct $\mathbf{\Sigma}$, and then compute the multivariate normal log-likelihood. Provide the value of the log-likelihood at these parameter values. Note: if `Sigma` is a matrix in `R`, then `det(Sigma)` returns the determinant of `Sigma` and `solve(Sigma)` returns the inverse of `Sigma`.

The log-likelihood is given by:

$$p(\boldsymbol{y}) = \frac{1}{\sqrt{2\pi}^m \sqrt{|\mathbf{\Sigma}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{y} - \mu \mathbf{1}_m)' \mathbf{\Sigma}^{-1} (\boldsymbol{y} - \mu \mathbf{1}_m)\right\}.$$

```
sig.sq.w <- 1
phi <- 0.25
mu <- 0
Sigma <- matrix(NA, nrow = m, ncol = m)
for (i in 1:nrow(Sigma)) {
  for (j in 1:nrow(Sigma)) {
    Sigma[i, j] <- phi^(abs(i - j))*sig.sq.w/(1 - phi^2)
  }
}
system.time(ll1 <- -log((2*pi)^m*det(Sigma))/2 - t(y[1:m])%*%solve(Sigma)%*%y[1:m]/2)
```

```
##    user  system elapsed
##   0.033   0.001   0.037
```

The log-likelihood at these parameter values is -261.13.

(b) Continuing to let $p = 1$, write down the likelihood for $y_1, \ldots, y_m$ as a simple function of the scalar mean μ, the autoregressive parameters ϕ_1, and the noise variance σ_w^2. Use the fact that we can write $p(\boldsymbol{y}) = p(y_1) \prod_{t=2}^m p(y_t | y_{t-1})$. Compute the log-likelihood for $\sigma_w^2 = 1$, $\phi_1 = 0.25$, and $\mu = 0$ using this way of writing down the log-likelihood. Provide the value of the log-likelihood at these parameter values.

The log-likelihood is given by:

$$p(y_1) \prod_{t=2}^m p(y_t | y_{t-1}) = \frac{1}{\sqrt{2\pi\sigma_w^2}^m \sqrt{1 - \phi_1^2}^{-1}} \exp\left\{-\frac{1}{2\sigma_w^2}\left((1 - \phi_1)^2 (y_1 - \mu)^2 + \sum_{t=2}^m (y_t - \mu - \phi(y_{t-1} - \mu))\right)\right\}.$$

```
system.time(ll2 <- -sum(log(2*pi*c(sig.sq.w/(1 - phi^2),
                                   rep(sig.sq.w, m - 1))))/2 -
              sum((y[1:m] - c(0, phi*y[1:(m-1)]))^2/c(sig.sq.w/(1 - phi^2),
```

```
                                                    rep(sig.sq.w,
                                                        m - 1)))/2)
```

```
##    user  system elapsed
##   0.000   0.000   0.001
```

The log-likelihood at these parameter values is -261.13.

(c) The `R` function `system.time` can be used to measure the amount of time running a line of `R` code takes. Using `system.time`, compare the amount of time it takes to compute the log-likelihood using the approaches described in (a) and (b). Comment on whether how the way we write the likelihood affects the time needed to compute the log-likelihood.

It takes much longer to evaluate the log-likelihood using the approach in (a) versus the approach in (b). This makes sense, because evaluating the log-likelihood using the approach in (a) requires inverting and taking the determinent of a 282×282 matrix, whereas evaluating the log-likelihood using the approach in (b) does not require any matrix inversions or determinents.

(d) Now let's fit some AR(p) models using `arima`. 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, $\phi_1, \ldots, \phi_p$ using `arima`. Make a plot with 6 panels. Using one panel for each value of p, plot the last 24 observations, the estimated fitted values from the corresponding fitted model $\widehat{\mathbb{E}}[y_t|y_{t-1}, \ldots, y_{t-p}] = \hat{\mu} + \sum_{i=1}^{p} \hat{\phi}_i (y_{t-i} - \hat{\mu})$, and the approximate 95% confidence intervals for the estimated fitted values. Note that approximate 95% confidence intervals can be computed from the standard deviations of $\widehat{\mathbb{E}}[y_t|y_{t-1}, \ldots, y_{t-p}] = \hat{\mu} + \sum_{i=1}^{p} \hat{\phi}_i (y_{t-i} - \hat{\mu})$, which can be computed from the p previous values $y_{t-1}, \ldots, y_{t-p}$ and the variance-covariance matrix of the estimated parameters returned by the `arima` function. If we set `armod <- arima(y[1:m], order = c(1, 0, 0))`, then `armod$var.coef` gives the variance-covariance matrix of the estimated parameters.

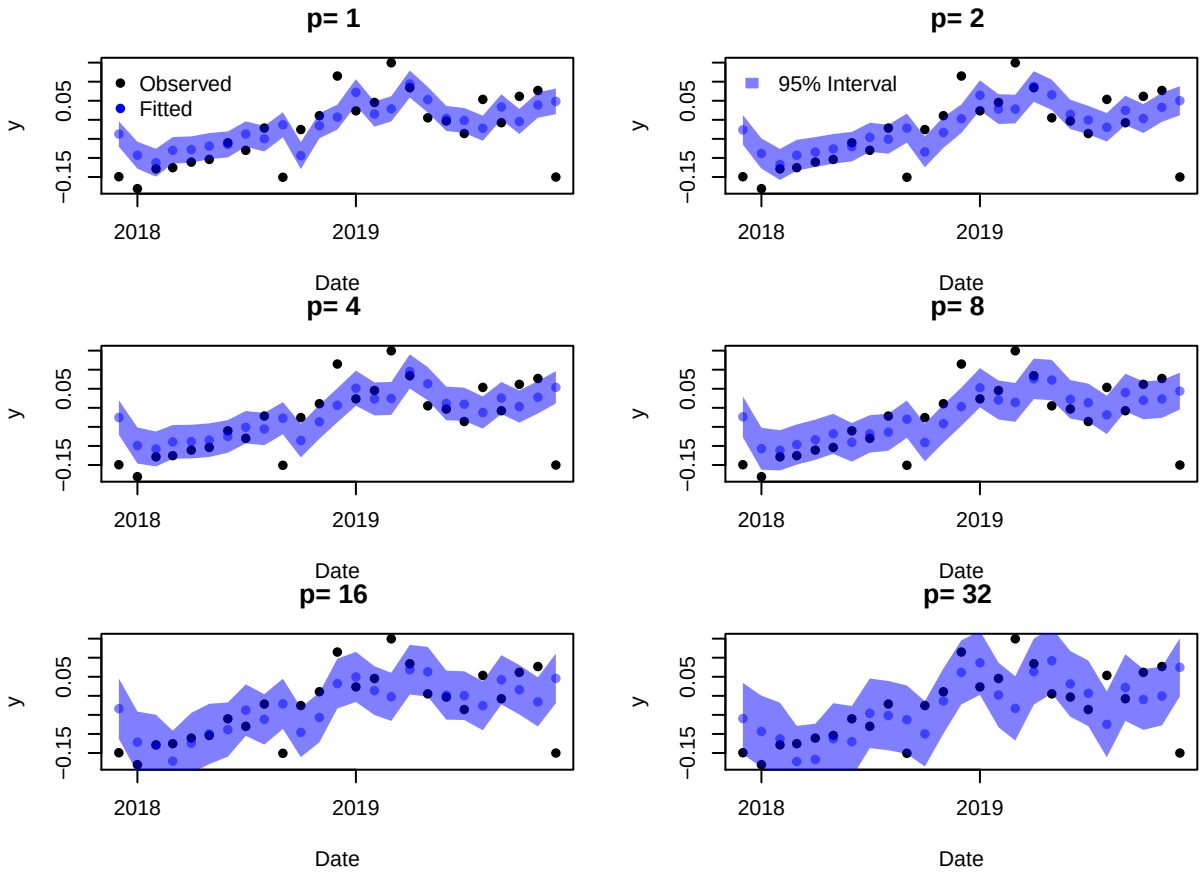
```
ps <- 2^(0:5)
ses <- fits <- array(NA, dim = c(length(y), length(ps)))

bics <- rep(length(ps))
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]
  for (i in 1:n) {
    if (i > p) {
      fits[i, which(p == ps)] <- mu.hat +
        sum(phi.hat*(y[(i - 1):(i - p)] - mu.hat))
      ses[i, which(p == ps)] <- sqrt(c(y[(i-p):(i-1)] - mu.hat,
                                       1)%*%armod$var.coef%*%c(y[(i-p):(i-1)] -
                                                               mu.hat, 1))
    }
  }

  bics[which(p == ps)] <- BIC(armod)
}
```

```
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$fdate[(length(y) - 24):length(y)],
       fits[, which(p == ps)][(length(y) - 24):length(y)],
       col = rgb(0, 0, 1, 0.5), pch = 16)
polygon(c(broc$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")
}
}
```



(e) Run the command `arima(y[1:36], order = c(15, 0, 0))`. Explain what this command is doing,

specifically the order of the model it is fitting and the number of observations (values of $\boldsymbol{y}$) which are being used. Comment on any errors you observe, and if you do observe any errors explain why they occur and how you could adjust the arguments of the `arima` function to obtain a solution.

This command is fitting an AR(p) model with $p = 15$ to the first 36 values of $\boldsymbol{y}$. When it is run, it produces the error `Error in arima(y[1:36], order = c(15, 0, 0)) : non-stationary AR part from CSS`. This error occurs because `R` first uses conditional sum of squares (CSS) to obtain starting values of the parameters μ, $\phi_1, \ldots, \phi_{15}$, and σ_w^2 for performing maximum likelihood. CSS fits the model to just $y_{16}, \ldots, y_{36}$, treating the problem as a standard regression problem where μ, $\phi_1, \ldots, \phi_{14}$ and ϕ_{15} are the regression coefficients and σ_w^2 is the noise variance. CSS can produce estimates of $\phi_1, \ldots, \phi_p$ that correspond to a *non-stationary* autoregressive model. However, the likelihood is not defined for values of $\phi_1, \ldots, \phi_p$ that correspond to a non-stationary model. For this reason, getting starting values for maximum likelihood by performing CSS first can fail. To fix this problem, we could add the argument `method='ML'` to the `arima` command, i.e. we would run `arima(y[1:36], order = c(15, 0, 0), method = 'ML')`. This will perform maximum likelihood starting at values of $\phi_1, \ldots, \phi_p$ that correspond to a stationary autoregressive model, but it may be slower because the starting values may not be close to the maximum likelihood estimates.