**ARFIMA models – Additional Examples**

Example: Varve data (varve\_frac.R)

The purpose here is to duplicate and expand on the results in a corresponding example of Shumway and Stoffer.

Varves are sedimentary deposits from melting glaciers over the spring and summer seasons. The data used here is a measurement of the thickness from yearly deposits at one location in Massachusetts. There are 634 years of deposit measurements that start about 11,800 years ago.

Below is part of the code and output for the analysis.

> library(package = astsa)

> y <- varve

> head(y)

[1] 26.28 27.42 42.28 58.28 20.57 28.57

> plot(x = y, ylab = expression(x[t]), xlab = "t",

 type = "l", col = "red", lwd = 1, main = "Glacial

 varve thickness data", panel.first = grid(col = "gray",

 lty = "dotted"))

> points(x = y, pch = 20, col = "blue")

 

y-axis label should be yt

The log transformation is applied to deal with nonstationarity in the variance.

> x <- log(y)

> plot(x = x, ylab = expression(log(x[t])), xlab =

 "t", type = "l", col = "red", lwd = 1, main =

 "Glacial varve thickness data - log transformed",

 panel.first = grid(col = "gray", lty = "dotted"))

> points(x = x, pch = 20, col = "blue")



> par(mfrow = c(1,2))

> acf(x = x, lag.max = 50, type = "correlation",

 main = "Est. ACF for log trans. data", xlim = c(1,50),

 ylim = c(-1,1))

> pacf(x = x, lag.max = 50, main = "Est. PACF for

 log trans. data", xlim = c(1,50), ylim = c(-1,1))

> par(mfrow = c(1,1))



> mean(x)

[1] 3.117993

> x.adj <- x-mean(x)

> mod.fit <- fracdiff(x = x.adj)

> mod.fit

Call:

 fracdiff(x = x.adj)

Coefficients:

 Estimate Std. Error z value Pr(>|z|)

d 3.705e-01 4.569e-06 81086 <2e-16 \*\*\*

---

Signif. codes:

0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

sigma[eps] = 0.4784262

[d.tol = 0.0001221, M = 100, h = 4.571e-06]

Log likelihood: -432.8 ==> AIC = 869.5471 [2 deg.freedom]

Notes:

* The ACF for xt is very slowly tailing off to 0.
* Notice that only d was estimated using fracdiff().
* The estimated ARFIMA model is (1-B)0.3705(xt-3.1179) = wt using the default value of M. Shumway and Stoffer found d = 0.384 with M = 30. To be consistent with Shumway and Stoffer’s original approach, I decided to use M = 30 and d = 0.384 in the rest of the analysis.
* Next, I decided to find the πj’s. Below is the code used and some of the resulting values.

> pi.vec <- numeric(length(x.adj))

> pi.vec[1] <- -mod.fit$d

> w <- numeric(length(x.adj))

> w[1] <- x.adj[1]

> for(j in 2:(length(x.adj))) {

 #Need to move subscript one back to match pi\_j+1

 equation

 pi.vec[j] <- ((j-1-mod.fit$d)\*pi.vec[j-1])/(j-1+1)

 w[j] <- x.adj[j] + sum(pi.vec[1:(j-1)]

 \* rev(x.adj[1:(j - 1)]))

 }

> pi.vec[1:5]

[1] -0.38416880 -0.11829157 -0.06371307 -0.04166566 –

 0.03013120

To find the πj’s, I used  and π0 = 1. The values above do not quite match those of Shumway and Stoffer. Their values are likely incorrect because they obtain π1 = 0.3079, but j = 0 produces

.

Thus, π1 = -d, which should be 0.384.

* The residuals can be obtained using the residuals() function. R does not automatically output the residuals.

> head(residuals(mod.fit))

[1] 0.1508147 0.1353409 0.5342287 0.6741805 -0.5507563

[6] 0.1059813

> tail(residuals(mod.fit))

[1] 0.131615807 -0.133828497 -0.655679563 0.724814316

[5] -0.002984594 -0.250338495

Let’s do the programming to find the residuals on our own too! Remember that the model can be written as

 = wt

⇔ π0xt + π1xt-1 + π2xt-2 + … = wt

where xt is already adjusted for the mean. The residuals are the wt’s resulting from the model’s fit. Note that

x1 = w1

x2+π1x1 = w2

x3+π1x2+π2x1 = w3

x634+π1x633+π2x632+…+π633x1 = w634

Below is a data frame containing the w’s

> all <- data.frame(x.adj, w, pi.vec)

> head(all)

 log.x.adj w pi.vec

1 0.15081473 0.1508147 -0.38416880

2 0.19327921 0.1353409 -0.11829157

3 0.62632070 0.5342287 -0.06371307

4 0.94726552 0.6741805 -0.04166566

5 -0.09415976 -0.5507563 -0.03013120

6 0.23436375 0.1059813 -0.02318009

> tail(all)

 x.adj w pi.vec

629 -0.19752367 0.131615807 -3.534658e-05

630 -0.46123656 -0.133828497 -3.526892e-05

631 -1.05620686 -0.655679563 -3.519155e-05

632 0.08597175 0.724814316 -3.511448e-05

633 -0.27424755 -0.002984594 -3.503769e-05

634 -0.56154165 -0.250338495 -3.496120e-05

These match what was obtained by residuals().

* Next are the residual ACF and PACF plots. The plot resembles an ACF plot from a white noise process. Remember that α = 0.05 here for each lag so it is not surprising to see a few values a little outside of the dashed lines.

> w <- residuals(mod.fit)

> par(mfrow = c(1,2))

> acf(x = w, lag.max = 100, type = "correlation", main =

 "Est. ACF for residuals", ylim = c(-1,1), panel.first =

 grid(col = "gray", lty = "dotted"))

> pacf(x = w, lag.max = 100, main = "Est. PACF for

 residuals", ylim = c(-1,1), panel.first =

 grid(col = "gray", lty = "dotted"))

> par(mfrow = c(1,1))



* How would you find ?



The mean needs to be added to this as well because x is mean adjusted. Also, we will need to put the forecasts back on the yt scale.

Forecasts in R:

> library(package = forecast)

> save.fore <- forecast(object = mod.fit, h = 2, level =

 95)

> save.fore

 Point Forecast Lo 95 Hi 95

635 -0.4145121 -1.353389 0.5243649

636 -0.3836371 -1.389413 0.6221390

> # Need to mean adjust

> fore.df <- data.frame(forecast = save.fore$mean +

 mean(x),

 low = as.numeric(save.fore$lower + mean(x)),

 up = as.numeric(save.fore$upper + mean(x)))

> # as.numeric() was needed to remove a label

> fore.df

 forecast low up

1 2.703481 1.764604 3.642358

2 2.734356 1.728580 3.740132

> # Need to account for log transform

> exp(fore.df)

 forecast low up

1 14.93162 5.839261 38.18178

2 15.39983 5.632652 42.10357

What would be the purpose of the forecasts here? Maybe a model like this would be helpful for current glaciers to understand varves.

* Shumway and Stoffer also examine regular ARIMA models for this data and suggest an ARIMA(1,1,1) model for the data. Later in the textbook they perform a “unit root test” to determine if d = 1. They conclude “no” and that the ARFIMA approach is better. Please see their textbook for the examples.

Example: ARFIMA(1,0.3,0) with ϕ1 = 0.2 and μ = 0 (arfima\_sim.R, x\_arfima.txt)

What if you did not know what the model was to begin with? I have not found any reference to “model building” strategies, but here is how I would proceed.

* Find the estimated ACF and check to see if it has the characteristics of a long-memory series.
* If it looks like a long-memory series, find an initial estimate of d using fracdiff().
* Find residuals from the model fit. Use the regular ARIMA model building methods to find estimates of ϕ’s and θ’s. Use fracdiff() to fit the models and update the estimate of d.

Following the above procedure, I fit an ARFIMA(0,d,0) model to the data, found the residuals, and then constructed an ACF plot of the residuals.

> mod.fit <- fracdiff(x = x.adj)

> summary(mod.fit)

Call:

 fracdiff(x = x.adj)

Coefficients:

 Estimate Std. Error z value Pr(>|z|)

d 3.760e-01 7.474e-06 50309 <2e-16 \*\*\*

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

[d.tol = 0.0001221, M = 100, h = 7.477e-06]

Log likelihood: -708.7 ==> AIC = 1421.316 [2 deg.freedom]

> w <- residuals(mod.fit)

> par(mfrow = c(1,2))

> acf(x = w, lag.max = 100, type = "correlation", main =

 "Est. ACF for residuals", ylim = c(-1,1),

 panel.first = grid(col = "gray", lty = "dotted"))

> pacf(x = w, lag.max = 100, main = "Est. PACF for

 residuals", ylim = c(-1,1), panel.first = grid(col

 = "gray", lty = "dotted"))

> par(mfrow = c(1,1))



The residual ACF and PACF look like the ACF and PACF from a white noise series. I was hoping that it would look like the ACF and PACF from an AR(1) process. This may have occurred due to the choice of a “small” ϕ1 value.

Suppose I wanted to find the residuals from the fit of an ARFIMA(1,d,0) model.

(1-Bϕ1)(1-B)dxt = wt

⇔ (1-Bϕ1)(xt + π1xt-1 + π2xt-2 + …) = wt

⇔ xt + π1xt-1 + π2xt-2 + …

- ϕ1xt-1 - ϕ1π1xt-2 - ϕ1π2xt-3 - … = wt

Thus,

x1 = w1

x2 + π1x1 - ϕ1x1 = w2

x3 + π1x2 + π2x1 - ϕ1x2 - ϕ1π1x1 = w3

x4 + π1x3 + π2x2 +π3x1 - ϕ1x3 - ϕ1π1x2 - ϕ1π2x1 = w4

x500+π1x499+π2x498+…+π499x1

- ϕ1x499 - ϕ1π1x498 - … - ϕ1π498x1 = w500

Below is the code used to find the residuals with this model. Also, residual ACF and PACF plots are produced.

> mod.fit <- fracdiff(x = x.adj, nar = 1)

> summary(mod.fit)

Call:

 fracdiff(x = x.adj, nar = 1)

Call:

 fracdiff(x = x.adj, nar = 1)

Coefficients:

 Estimate Std. Error z value Pr(>|z|)

d 0.250143 0.007397 33.816 < 2e-16 \*\*\*

ar 0.187409 0.044551 4.207 2.59e-05 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

sigma[eps] = 0.9928298

[d.tol = 0.0001221, M = 100, h = 7.45e-06]

Log likelihood: -706.1 ==> AIC = 1418.169 [3 deg.freedom]

> w <- residuals(mod.fit)

> par(mfrow = c(1,2))

> acf(x = w, lag.max = 100, type = "correlation", main =

 "Est. ACF for residuals", ylim = c(-1,1), panel.first =

 grid(col = "gray", lty = "dotted"))

> pacf(x = w, lag.max = 100, main = "Est. PACF for

 residuals", ylim = c(-1,1), panel.first = grid(col

 = "gray", lty = "dotted"))

> par(mfrow = c(1,1))



Again, the residual ACF and PACF plots look similar to the same plots from a white noise process.

Examine the AIC to choose between the models. The AIC for the model that includes ϕ1 is a little lower.

Final notes:

* If a good estimate for d can be found, then (1-B)dxt can be found. An ARMA could be built upon this series using arima() and tsdiag(). I have not tried this and I do not know how well this would work.
* The arfima package can also estimate ARFIMA models. Below is output associated with the ARFIMA(1,0.3,0) model example

> mod.fit2 <- arfima(z = x, order = c(1, 0, 0))

Note: only one starting point. Only one mode can be found -- this is now the default behavior.

Beginning the fits with 1 starting values.

> summary(mod.fit2)

Call:

arfima(z = x, order = c(1, 0, 0))

Mode 1 Coefficients:

 Estimate Std. Error Th. Std. Err. z-value Pr(>|z|)

phi(1) 0.1860171 0.0838391 0.0828130 2.21874 0.0265046 \*

d.f 0.2513300 0.0664605 0.0657191 3.78164 0.0001558 \*\*\*

Fitted mean -0.1457951 0.2662330 NA -0.54762 0.5839513

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

sigma^2 estimated as 0.988215; Log-likelihood = 3.61017; AIC = 0.779664; BIC = 17.6381

Numerical Correlations of Coefficients:

 phi(1) d.f Fitted mean

phi(1) 1.00 -0.85 -0.03

d.f -0.85 1.00 0.03

Fitted mean -0.03 0.03 1.00

Theoretical Correlations of Coefficients:

 phi(1) d.f

phi(1) 1.00 -0.85

d.f -0.85 1.00

Expected Fisher Information Matrix of Coefficients:

 phi(1) d.f

phi(1) 1.04 1.11

d.f 1.11 1.64

The model is essentially the same as before. The largest difference is the estimate of μ is done within the function and it is a little more than what we obtained previously.

Forecasts:

> save.pred2 <- predict(object = mod.fit2, n.ahead = 2,

 predint = 0.95)

> save.pred2

$`Mode 1`

$`Mode 1`$`Forecasts and SDs`

 1 2

Forecasts -0.683843 -0.500583

Exact SD 0.994153 1.085122

Limiting SD 0.994090 1.085004

> names(save.pred2)

[1] "" "z" "limiting" "predint" "name" "m"

> save.pred2$"" # Oddly, the first component in the list

 does not have a name!

NULL

> save.pred2[[1]] # So, access by number

$SDForecasts

 1 2

500 0.9941527 1.085122

$Forecast

 1 2

-0.6838431 -0.5005828

$exactVar

[1] 0.9883396 1.1774905

$exactSD

[1] 0.9941527 1.0851224

$limitVar

[1] 0.9882149 1.1772333

$limitSD

[1] 0.994090 1.085004

$sigma2

[1] 0.9882149

> low <- save.pred2[[1]]$Forecast - qnorm(p = 0.975, mean

 = 0, sd = 1) \* save.pred2[[1]]$limitSD

> up <- save.pred2[[1]]$Forecast + qnorm(p = 0.975, mean =

 0, sd = 1) \* save.pred2[[1]]$limitSD

> data.frame(low, up)

 low up

1 -2.632224 1.264538

2 -2.627151 1.625986

There are some small differences in comparison to fracdiff.

> plot(x = save.pred2)



Why didn’t we focus on this package?

* The fracdiff package is more established. It has been around since at least 1999 in R and available with S-Plus prior to it.
* I recognize some of fracdiff’s authors as leaders in time series analysis.

Both packages could use improvements.