**ARFIMA models – Introduction and Simulated Example**

The purpose of this section is to give a brief account of long memory ARMA models. The “F” in ARFIMA stands for “fractional” with respect to differencing.

ARIMA(p,0,q)(P,0,Q)s is referred to as a “short memory” process because ρ(h)→0 “quickly” as h→∞.

A long memory time series does not and has ρ(h)→0 “slowly”.

In the past, we would have differenced a time series that exhibited these properties. There are cases where using (1-B)d for d = 1, 2, … may be too severe to fix the problem and result in “overdifferencing”. Note that overdifferencing may induce artificial patterns in the time series and reduce forecasting accuracy (see Pankratz’s textbook).

Long memory time series do not necessarily have “large” ρ(h). Instead, the ρ(h) ≠ 0 tend to persist for a long period of time. Some of the original examples where this type of data was examined include economics and hydrology applications from the 1980s. Newer examples include computer network traffic and meteorology.

As a compromise between differencing and not differencing a time series, fractional differencing can be done.

Use fractional powers of d between 0 to 0.5 for the differencing. The fractional difference series (p = 0, q = 0) can be represented as

(1-B)dxt =  = wt

where wt ~ ind. (0,) and the πj’s are found through a Taylor series expansion of (1-B)d. Note that d now becomes a parameter to be estimated!

The πj’s

Taylor series expansion in general for a function f(B):

f(B) =  where f(j)(a) is the jth derivative of f(B) evaluated at a

Using a Taylor series expansion about 0 leads to



The general expression is

 (1-B)d =  where 

Notes:

* Γ(x+1) = xΓ(x) = x for x > 0 is the gamma function. Note that if x is an integer, this simplifies to Γ(x) = (x – 1)!
	+ - How can I have Γ(-d) above???
		- Notice the Γ(-d) will fall out; for example,



* An equivalent recursive representation of πj is

 and π0 = 1

One can try a few values of j to see it works out.

* It can be shown that  where



Finding estimate of d:

Use an iterative root-finding procedure with Q(d) = . In other words, minimize the squared wt’s (residuals) with respect to d. Iterative numerical method details are available in Shumway and Stoffer’s textbook.

Fractional ARIMA(p,d,q) or ARFIMA model

The model is

ϕ(B)(1-B)d(xt-μ) = θ(B)wt

for 0 < d < 0.5 and wt ~ ind. (0,). This model can be rewritten as

ϕ(B)πd(B)(xt-μ) = θ(B)wt

where πd(B) = .

Notes:

* A seasonal form of the model could also have been used above.
* We have often just examined the μ = 0 form of an ARMA model. The way we have written the μ ≠ 0 form of the model is ϕ(B)xt = α + θ(B)wt where α = μ(1-ϕ1-ϕ2-…-ϕp). Instead, one could also write the model as ϕ(B)(xt-μ) = θ(B)wt so that the α term is not directly written in there. This is why the ARFIMA model is represented as above.
* Let zt = (1-B)dxt. Thus, xt is passed through an infinite linear filter to form zt. We can then find an ARMA model for zt.
* The wt’s are still assumed to be independent N(0,) random variables when estimating the model. This can be used the “usual” way for diagnostic methods.
* Parameter estimation and forecasting can proceed using similar techniques as described earlier in the course!
* The fracdiff package in R provides the computational methods.

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

To simulate data from an ARFIMA model, use the fracdiff.sim() function in R. Below is part of the code and output.

> library(fracdiff)

> set.seed(9101)

> x.ex <- fracdiff.sim(n = 500, ar = 0.2, ma = NULL, d =

 0.3, rand.gen = rnorm, sd = 1, mu = 0)

> names(x.ex)

[1] "series" "ar" "ma" "d" "mu"

[6] "n.start"

> x.ex$series[1:5]

[1] 0.2558553628 -0.2582559302 1.1313066446 0.0001357745

[6] 1.1763489841

I had already simulated observations from the model in the past and put them in x\_arfima.txt. This data was simulated using the same settings as above, but just using a different seed. I am going to use this data for the rest of the example. Below is the code needed to read in the data and plot the data.

> arfima.sim.data <- read.table(file = "x\_arfima.txt",

 header = TRUE, sep = "")

> head(arfima.sim.data)

 x

1 1.3563372

2 1.5681298

3 0.8548125

4 1.1731924

5 0.7413035

6 0.7017789

> x <- arfima.sim.data$x

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

 "l", col = "red", lwd = 1, main = "Plot of x\_arfima.txt

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

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



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

> acf(x = x, type = "correlation", main = "Estimated ACF

 for x\_arfima.txt data", ylim = c(-1,1), lag.max = 50)

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

 x\_arfima.txt data", ylim = c(-1,1))

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



The ACF of the simulated data is tailing off very slowly – although the autocorrelations are not very large. This is a characteristic of a long-memory series. Note that this particular series has autocorrelations tailing off faster than most others that I have generated! Try the code yourself to see other examples of the ACF.

To fit a model to the data, the fracdiff() function can be used. The data should be first adjusted to make sure it has a 0 mean. Below is part of the code and output.

> mean(x)

[1] -0.1712369

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

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

> #use nma = 1 for 1 MA if needed

> mod.fit

Call:

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

Coefficients:

 d ar

0.2501425 0.1874089

sigma[eps] = 0.9928298

a list with components:

 [1] "log.likelihood" "n" "msg"

 [4] "d" "ar" "ma"

 [7] "covariance.dpq" "fnormMin" "sigma"

[10] "stderror.dpq" "correlation.dpq" "h"

[13] "d.tol" "M" "hessian.dpq"

[16] "length.w" "residuals" "fitted"

[19] "call"

> summary(mod.fit)

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]

> vcov(mod.fit)

 d ar1

d 5.471673e-05 -0.0000589026

ar1 -5.890260e-05 0.0019848122

> names(mod.fit)

 [1] "log.likelihood" "n" "msg"

 [4] "d" "ar" "ma"

 [7] "covariance.dpq" "fnormMin" "sigma"

[10] "stderror.dpq" "correlation.dpq" "h"

[13] "d.tol" "M" "hessian.dpq"

[16] "length.w" "residuals" "fitted"

[19] "call"

Notes:

* The estimated model is

(1 - 0.1875B)(1 - B)0.2501(xt + 0.1712) = wt.

with . Notice how the mean is incorporated into the model.

* A test of Ho:ϕ1=0 vs. Ha:ϕ1≠0 results in



Thus, there is sufficient evidence to show that ϕ1 ≠ 0.

* Similarly, there is sufficient evidence to show d ≠ 0.
* Computational speed has improved since the 1980s when these models were developed! Haslett and Raftery’s 1989 paper discuss using a VAX computer to do the calculations. They say a "single evaluation of the likelihood takes 3 hours of CPU time on a VAX 11/780 and finding the MLE would take at least 45 hours.”
* To speed up the computational process, Haslett and Raftery use an algorithm that omits M initial observations in the computations. This algorithm is implemented by fracdiff() using M = 100 as the default. Page 13 of their paper provides some reasoning (approximations are made using the first 100 observations to help with the calculations using the remaining observations). With their algorithm, a single evaluation took 2.5 minutes.

How would you find ? Remember that an ARFIMA model can be written in general as

ϕ(B)πd(B)(xt-μ) = θ(B)wt

where πd(B) = . For our model, let zt = xt - μ. Then





This leads to



After recording the video: The equation below has been corrected for two errors.

The mean needs to be added to this as well because z is mean adjusted. Thus,



Unfortunately, the fracdiff package does not have a function available to find the forecasts. Below is code that can be used instead.

> library(package = forecast)

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

 95)

> save.fore

 Point Forecast Lo 95 Hi 95

501 -0.5184022 -2.465437 1.428633

502 -0.3372291 -2.462489 1.788030

> names(save.fore)

 [1] "x" "mean" "upper" "lower"

 [5] "level" "method" "model" "series"

 [9] "residuals" "fitted"

> save.fore$mean

Time Series:

Start = 501

End = 502

Frequency = 1

[1] -0.5184022 -0.3372291

> save.fore$mean + mean(x)

Time Series:

Start = 501

End = 502

Frequency = 1

[1] -0.6896391 -0.5084660

> save.fore$upper + mean(x)

Time Series:

Start = 501

End = 502

Frequency = 1

 95%

[1,] 1.257396

[2,] 1.616793

> save.fore$lower + mean(x)

Time Series:

Start = 501

End = 502

Frequency = 1

 95%

[1,] -2.636674

[2,] -2.633725

> plot(save.fore, xlim = c(495, 503))

