**ARCH Models – Simulated Example**

There are a number of R packages available to work with these models. Packages include:

1. tseries and the garch() function
2. fGarch and the garchFit() function
3. rugarch and the ugarchfit() function

The finance task view at CRAN (<http://cran.r-project.org/web/views/Finance.html>) gives a summary of these and other packages available for finance data modeling.

The tseries package has limitations for what can be done with the model so we will focus on fGarch. The rugarch package works fine as well, so some examples will be provided with it. Code for the tseries package is available my corresponding programs when it will work for an example.

Example: Generate data from an ARCH( 1) model with α0 = 0.1 and α1 = 0.4 (arch1.R)

Code to simulate data from the model:

set.seed(1532)

n <- 1100

a <- c(0.1, 0.4) *#ARCH(1) coefficients - alpha0 and alpha1*

e <- rnorm(n = n, mean = 0, sd = 1)

y <- numeric(n) *#intializes a vector of y's to be n long*

y[1] <- rnorm(n = 1, mean = 0, sd = sqrt(a[1]/(1.0-a[2])))

 *#start value*

for(i in 2:n) *#Simulate ARCH(1) process*

 {

 y[i] <- e[i]\*sqrt(a[1] + a[2]\*y[i-1]^2)

 }

y <- y[101:1100] *#Drop the first 100 and just call it y*

 *again*

save.y <- y

The start value for y1 needs to be set outside the loop. The variance used is Var(yt) = α0/(1-α1) as found previously in the notes.

Below is a plot of the data. The plot of yt shows moments of high volatility in comparison to other time points.

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

 "l", col = "red", main = "ARCH(1) simulaed data",

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

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



While I used “y” here, we would actually find a model for the mean adjusted version of it.

This data could have been generated more easily by using the garchSim()function in the fGarch package.

> library(fGarch)

> #Note that beta (sigma^2\_t-1 part) needs to be set to

 something due to a default value of 0.8

> set.seed(9129)

> spec <- garchSpec(model = list(omega = 0.1, alpha = 0.4,

 beta = 0))

> x <- garchSim(spec = spec, n = 1000)

> head(x)

GMT

 garch

2019-04-16 -0.33502258

2019-04-17 -0.01923007

2019-04-18 -0.27130332

2019-04-19 -0.06068358

2019-04-20 0.11332049

2019-04-21 0.51447406

> tail(x)

GMT

 garch

2022-01-04 0.15444146

2022-01-05 0.50981301

2022-01-06 0.06590522

2022-01-07 0.28678420

2022-01-08 0.07107253

2022-01-09 -0.69714833

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

 "l", col = "red", main = "ARCH(1) simulated data",

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

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



Notes:

* I am unable to remove the date information produced by garchSim().
* The garchSpec() model specification can be generalized for other versions of a GARCH model. For example, an ARCH(2) model uses the alpha argument to specify α1 and α2 (use c()).

In a typical model building situation when you do not know if an ARIMA and/or ARCH model is appropriate, one should find the ACF and PACF for xt and .

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

> acf(x = x, type = "correlation", main = "Est. ACF for

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

 lty = "dotted"))

> pacf(x = x, main = "Est. PACF for x", ylim = c(-1,1),

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



There is not any strong indication of dependence among xt for t = 1, …, n. This indicates that an ARIMA model is likely not needed.

> acf(x = x^2, type = "correlation", main =

 expression(paste("Est. ACF for ", x^2)), ylim =

 c(-1,1), panel.first = grid(col = "gray", lty =

 "dotted"))

> pacf(x = x^2, main = expression(paste("Est. PACF for ",

 x^2)), ylim = c(-1,1), panel.first = grid(col =

 "gray", lty = "dotted"))

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



There are significant ACF and PACF values for . The patterns in these plots are similar to those for an AR(1). Therefore, an ARCH(1) model should be investigated.

> mod.fit <- garchFit(formula = ~ garch(1, 0), data = x,

 include.mean = TRUE)

Series Initialization:

 ARMA Model: arma

 Formula Mean: ~ arma(0, 0)

<OUTPUT EDITED>

> summary(mod.fit)

Title:

 GARCH Modelling

Call:

 garchFit(formula = ~garch(1, 0), data = x)

Mean and Variance Equation:

 data ~ garch(1, 0)

<environment: 0x000000000e6a7528>

 [data = x]

Conditional Distribution:

 norm

Coefficient(s):

 mu omega alpha1

0.00041282 0.09665398 0.42442295

Std. Errors:

 based on Hessian

Error Analysis:

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

mu 0.0004128 0.0108266 0.038 0.97

omega 0.0966540 0.0066149 14.612 < 2e-16 \*\*\*

alpha1 0.4244229 0.0608758 6.972 3.13e-12 \*\*\*

---

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

Log Likelihood:

 -452.1397 normalized: -0.4521397

Standardised Residuals Tests:

 Statistic p-Value

 Jarque-Bera Test R Chi^2 0.4202807 0.8104705

 Shapiro-Wilk Test R W 0.9981939 0.3735418

 Ljung-Box Test R Q(10) 15.1952 0.1251057

 Ljung-Box Test R Q(15) 18.0511 0.2599886

 Ljung-Box Test R Q(20) 20.52373 0.4256256

 Ljung-Box Test R^2 Q(10) 3.841834 0.9542001

 Ljung-Box Test R^2 Q(15) 14.25011 0.5066509

 Ljung-Box Test R^2 Q(20) 15.26702 0.7609304

 LM Arch Test R TR^2 5.875025 0.9222503

Information Criterion Statistics:

 AIC BIC SIC HQIC

0.9102794 0.9250027 0.9102615 0.9158753

Notes:

* The garchFit()function fits the model. The order of the model is given as (m,r) where m is the order of the ARCH part and r is order of the GARCH part. **BE CAREFUL** because textbooks and software are not consistent in their orderings.
* Use trace = FALSE to reduce the amount of information given when running garchFit().
* A warning message is given when running garchFit():

After recording the video: The warning no longer occurs. The authors have updated their code.

Warning message:

Using formula(x) is deprecated when x is a character vector of length > 1.

 Consider formula(paste(x, collapse = " ")) instead.

Code within the function itself needs to be updated by its authors.

* The model for the data is

xt = 0.0004128 + wt

with wt ~ N(0, , , and yt = xt - 0.0004128. Equivalently, we could state the model as

 = εt

where , yt = xt – 0.0004128, and εt ~ N(0,1).

* Notice how close the parameter estimates are to the true parameters (use the standard error to help measure closeness).
* The standard tests for whether or not the α0 = 0 or α1 = 0 are given in the coefficients table of the output. Both are significantly different from 0 as would be expected. Even though the output says “t value” and “Pr(>|t|)”, a normal distribution approximation is made to the sampling distribution of the test statistic.
* The Ljung-Box-Pierce test is the same test as discussed before. This test is performed upon the residuals (R) and squared residuals (R^2) for lags 10, 15, and 20. All of the p-values are larger indicating there is no dependence remaining in the residuals or squared residuals.
* The Jarque-Bera and Shapiro-Wilk tests are tests for normality of the residuals (null hypothesis is a normal distribution). The large p-values suggest there is not sufficient evidence against the normality assumption.
* A model can be fit to the original simulated data as well. The estimated model is



with . The code for this estimation is within the program.

To view what is inside of mod.fit, we need to use the slotNames() function rather than the usual names() function.

Most of R and its corresponding packages are written in a form very similar to the S programming language. This language was first developed in the 1970s at Bell Laboratories with its main designer being John Chambers. Version 3 of S (S3) is emulated most by R, and this is what was used primarily before this example. Version 4 (S4) is used by the fGarch and rugarch packages. The components of an object in S4 are called “slots”. To access a slot, use the syntax <object name>@<slot name>.

> names(mod.fit)

NULL

> slotNames(mod.fit)

 [1] "call" "formula" "method" "data"

 [5] "fit" "residuals" "fitted" "h.t"

 [9] "sigma.t" "title" "description"

> tail(mod.fit@fitted)

 2021-11-24 2021-11-25 2021-11-26 2021-11-27

0.0004128227 0.0004128227 0.0004128227 0.0004128227

 2021-11-28 2021-11-29

0.0004128227 0.0004128227

> tail(mod.fit@residuals)

[1] 0.15402864 0.50940019 0.06549239 0.28637137

[5] 0.07065971 -0.69756116

> tail(x - mod.fit@fitted)

GMT

 garch

2011-10-17 0.15402864

2011-10-18 0.50940019

2011-10-19 0.06549239

2011-10-20 0.28637137

2011-10-21 0.07065971

2011-10-22 -0.69756116

> tail(residuals(object = mod.fit))

[1] 0.15402864 0.50940019 0.06549239 0.28637137

[5] 0.07065971 -0.69756116

> tail(mod.fit@sigma.t)

[1] 0.4175212 0.3266854 0.4547382 0.3138064 0.3625745

[6] 0.3142818

> sqrt(0.0966540+0.4244229\*(x[999]-0.0004128)^2)

[1] 0.3142818

> tail(mod.fit@h.t) #sigma.t^2 for us

[1] 0.17432393 0.10672334 0.20678688 0.09847444 0.13146029

[6] 0.09877303

> tail(mod.fit@sigma.t^2)

[1] 0.17432393 0.10672334 0.20678688 0.09847444 0.13146029

[6] 0.09877303

> mod.fit@fit$matcoef[,1]

 mu omega alpha1

0.0004128227 0.0966539778 0.4244229495

> e <- (x - mod.fit@fit$matcoef[1,1])/mod.fit@sigma.t

> tail(e)

GMT

 garch

2011-10-17 0.3689122

2011-10-18 1.5592990

2011-10-19 0.1440222

2011-10-20 0.9125735

2011-10-21 0.1948833

2011-10-22 -2.2195406

> tail(residuals(object = mod.fit, standardize = TRUE))

[1] 0.3689122 1.5592990 0.1440222 0.9125735 0.1948833

[6] -2.2195406

> # Can use residuals(object = mod.fit, standardize = TRUE) > # for e

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

> acf(x = e, type = "correlation", lag.max = 20, ylim =

 c(-1,1), xlab = "h", main = "ARCH residual ACF")

> pacf(x = e, lag.max = 20, ylim = c(-1,1), xlab = "h",

 main = "ARCH residual PACF")

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



Notes:

* Notice the “@fitted” values are all equal to  Why?
* The “@residuals” values are the observed time series minus .
* I found estimates of the residuals for the ARCH part of the model as et = /. You can also obtain these as well by using the residuals() function with the standardize = TRUE argument value.
* The ACF and PACF plots (not shown above) for et do not show any significant autocorrelations or partial autocorrelations as expected.

The plot() function with the model fitting object leads to a number of plots that can be produced:

> plot(mod.fit)

Make a plot selection (or 0 to exit):

 1: Time Series

 2: Conditional SD

 3: Series with 2 Conditional SD Superimposed

 4: ACF of Observations

 5: ACF of Squared Observations

 6: Cross Correlation

 7: Residuals

 8: Conditional SDs

 9: Standardized Residuals

10: ACF of Standardized Residuals

11: ACF of Squared Standardized Residuals

12: Cross Correlation between r^2 and r

13: QQ-Plot of Standardized Residuals

Selection:

Enter an item from the menu, or 0 to exit

Selection: 1



Make a plot selection (or 0 to exit):

 1: Time Series

 2: Conditional SD

 3: Series with 2 Conditional SD Superimposed

 4: ACF of Observations

 5: ACF of Squared Observations

 6: Cross Correlation

 7: Residuals

 8: Conditional SDs

 9: Standardized Residuals

10: ACF of Standardized Residuals

11: ACF of Squared Standardized Residuals

12: Cross Correlation between r^2 and r

13: QQ-Plot of Standardized Residuals

Selection: 5



Make a plot selection (or 0 to exit):

 1: Time Series

 2: Conditional SD

 3: Series with 2 Conditional SD Superimposed

 4: ACF of Observations

 5: ACF of Squared Observations

 6: Cross Correlation

 7: Residuals

 8: Conditional SDs

 9: Standardized Residuals

10: ACF of Standardized Residuals

11: ACF of Squared Standardized Residuals

12: Cross Correlation between r^2 and r

13: QQ-Plot of Standardized Residuals

Selection: 10



Make a plot selection (or 0 to exit):

 1: Time Series

 2: Conditional SD

 3: Series with 2 Conditional SD Superimposed

 4: ACF of Observations

 5: ACF of Squared Observations

 6: Cross Correlation

 7: Residuals

 8: Conditional SDs

 9: Standardized Residuals

10: ACF of Standardized Residuals

11: ACF of Squared Standardized Residuals

12: Cross Correlation between r^2 and r

13: QQ-Plot of Standardized Residuals

Selection: 11



Make a plot selection (or 0 to exit):

 1: Time Series

 2: Conditional SD

 3: Series with 2 Conditional SD Superimposed

 4: ACF of Observations

 5: ACF of Squared Observations

 6: Cross Correlation

 7: Residuals

 8: Conditional SDs

 9: Standardized Residuals

10: ACF of Standardized Residuals

11: ACF of Squared Standardized Residuals

12: Cross Correlation between r^2 and r

13: QQ-Plot of Standardized Residuals

Selection:

Enter an item from the menu, or 0 to exit

Selection: 13



To produce just one plot, you can use

plot(mod.fit, which = 1)

where the which argument corresponds to the plot number seen in the list.

Forecasting can be done using the predict() function:

> # Forecasts

> predict(object = mod.fit, n.ahead = 3, plot = TRUE, nx =

 3, conf = 0.95)

 meanForecast meanError standardDeviation lowerInterval

1 0.0004128227 0.5506129 0.5506129 -1.0787687

2 0.0004128227 0.4746875 0.4746875 -0.9299576

3 0.0004128227 0.4385071 0.4385071 -0.8590452

 upperInterval

1 1.0795943

2 0.9307833

3 0.8598709



After recording the video: m here represents the number of time points into the future for the forecast, not the order of the GARCH model

How are these forecasts found?

We started with xt. There are no ϕi or θj parameters in the model so forecasted values are μ. We also set yt = xt – μ. Note that  = E(yn+m | In) = 0. Thus, the forecasted value for xn+m is  and for yn+m is 0. The variance needed for a CI given by



because of the simplicity of the model. Then



because



Our model for  is  = α0 + α1. Thus,



Substituting the parameter estimates and n = 1000, we obtain

> 0.0966540 + 0.4244229\*(x[1000]-0.0004128)^2

[1] 0.3031746

for  or equivalently , which matches the “standardDeviation” column in the output.

The (1 - α)100% C.I. for xn+m is



Substituting the parameter estimates and n = 1000,

> sigma.1001 <- sqrt(0.0966540 + 0.4244229\*(x[1000]

 - 0.0004128)^2)

> 0.0004128 - qnorm(p=0.975)\*sigma.1001

[1] -1.078769

> 0.0004128 + qnorm(p=0.975)\*sigma.1001

[1] 1.079594

which matches the output.

For m = 2, we have . We saw earlier in the course notes that  = α0 + α1. Using this result, we can write  as



We find  and the 95% confidence interval for x1002 to be

> sigma.1002 <- sqrt(0.0966540 + 0.4244229 \*

 sigma.1001^2)

> sigma.1002

[1] 0.4746875

> 0.0004128-qnorm(p=0.975)\*sigma.1002

[1] -0.9299576

> 0.0004128 + qnorm(p=0.975)\*sigma.1002

[1] 0.9307832

which matches the output.

The rugarch package’s data simulation function ugarchsim() needs to use a model fitting object from ugarchfit() to simulate data. Below is an example of how to fit a model to the original simulated data (save.y) and then to use the resulting model fit object to simulate data.

> spec <- ugarchspec(variance.model = list(model =

 "sGARCH", garchOrder = c(1,0)), mean.model =

 list(armaOrder = c(0, 0), include.mean = TRUE, arfima =

 FALSE))

> mod.fit <- ugarchfit(spec = spec, data = save.y)

> summary(mod.fit)

 Length Class Mode

 1 uGARCHfit S4

> show(mod.fit)

\*---------------------------------\*

\* GARCH Model Fit \*

\*---------------------------------\*

Conditional Variance Dynamics

-----------------------------------

GARCH Model : sGARCH(1,0)

Mean Model : ARFIMA(0,0,0)

Distribution : norm

Optimal Parameters

------------------------------------

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

mu -0.013252 0.010990 -1.2059 0.22785

omega 0.099077 0.006358 15.5821 0.00000

alpha1 0.355799 0.052955 6.7189 0.00000

Robust Standard Errors:

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

mu -0.013252 0.011018 -1.2028 0.22904

omega 0.099077 0.006581 15.0556 0.00000

alpha1 0.355799 0.067837 5.2449 0.00000

LogLikelihood : -433.8668

Information Criteria

------------------------------------

Akaike 0.87373

Bayes 0.88846

Shibata 0.87372

Hannan-Quinn 0.87933

Weighted Ljung-Box Test on Standardized Residuals

------------------------------------

 statistic p-value

Lag[1] 2.677 0.1018

Lag[2\*(p+q)+(p+q)-1][2] 2.745 0.1645

Lag[4\*(p+q)+(p+q)-1][5] 3.807 0.2791

d.o.f=0

H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals

------------------------------------

 statistic p-value

Lag[1] 2.354 0.1250

Lag[2\*(p+q)+(p+q)-1][2] 2.557 0.1851

Lag[4\*(p+q)+(p+q)-1][5] 3.015 0.4044

d.o.f=1

Weighted ARCH LM Tests

------------------------------------

 Statistic Shape Scale P-Value

ARCH Lag[2] 0.4045 0.500 2.000 0.5248

ARCH Lag[4] 0.7954 1.397 1.611 0.7722

ARCH Lag[6] 0.9093 2.222 1.500 0.9118

Nyblom stability test

------------------------------------

Joint Statistic: 0.6268

Individual Statistics:

mu 0.26102

omega 0.09566

alpha1 0.18167

Asymptotic Critical Values (10% 5% 1%)

Joint Statistic: 0.846 1.01 1.35

Individual Statistic: 0.35 0.47 0.75

Sign Bias Test

------------------------------------

 t-value prob sig

Sign Bias 2.7735 0.005649 \*\*\*

Negative Sign Bias 1.4811 0.138889

Positive Sign Bias 0.6937 0.488002

Joint Effect 8.6820 0.033831 \*\*

Adjusted Pearson Goodness-of-Fit Test:

------------------------------------

 group statistic p-value(g-1)

1 20 21.44 0.3130

2 30 36.32 0.1644

3 40 36.72 0.5743

4 50 42.70 0.7250

Elapsed time : 0.107878

Notes:

* The model specification is different from what we have seen so far. Notice the arfima = FALSE argument value. If one wanted to include an ARFIMA model for the original series, this where you can say TRUE. Notice the output says “ARFIMA(0,0,0)” model, but a regular ARIMA model is used unless the arfima argument is TRUE.
* Notice the use of show() rather than summary() to obtain a summary of the model fit.
* The ARCH(1) model is  = εt where  and . These estimates are VERY similar to those produced by garchFit().

Below is an example of simulating new data from this model:

> set.seed(1828)

> x.sim <- ugarchsim(fit = mod.fit, n.sim = 1000)

> slotNames(x.sim)

[1] "simulation" "model" "seed"

> head(x.sim@simulation$seriesSim)

 [,1]

[1,] 0.293186348

[2,] 0.165803842

[3,] -0.721580395

[4,] -0.223599151

[5,] -0.005500317

[6,] 0.643013179

When I try to fit the corresponding model to the data, the model does not converge! I was successful with other simulated data.