Univariate Time Series with R

Tyler J. Brough

November 16, 2016

Using the Quantmod Package to Retrieve Financial Time Series

First, make sure you have `quantmod` installed:

```
suppressMessages(require("quantmod"))
```

If it is not, you will want to install it with:

```
$ install.packages("quantmod")
```

Once we have the package installed we can use it to pull historical stock price data. Let's use it to get some stock price data for ticker MSFT from Google Finance:

```
getSymbols('MSFT', src = "google")
```

```
##     As of 0.4-0, 'getSymbols' uses env=parent.frame() and
##  auto.assign=TRUE by default.
##
##  This  behavior  will be  phased out in 0.5-0  when the call  will
##  default to use auto.assign=FALSE. getOption("getSymbols.env") and
##  getOptions("getSymbols.auto.assign") are now checked for alternate defaults
##
##  This message is shown once per session and may be disabled by setting
##  options("getSymbols.warning4.0"=FALSE). See ?getSymbols for more details.
```

```
## [1] "MSFT"
```

The data will be stored in an `xts` object, which we can inspect as follows:

```
data.class(MSFT)
```

```
## [1] "xts"
```

```
head(MSFT)
```

```
##            MSFT.Open MSFT.High MSFT.Low MSFT.Close MSFT.Volume
## 2007-01-03     29.91     30.25    29.40      29.86    77574283
## 2007-01-04     29.70     29.97    29.44      29.81    46120855
## 2007-01-05     29.63     29.75    29.45      29.64    44677778
## 2007-01-08     29.65     30.10    29.53      29.93    50226020
## 2007-01-09     30.00     30.18    29.73      29.96    44677271
## 2007-01-10     29.80     29.89    29.43      29.66    55048885
```

```
tail(MSFT)
```

```
##            MSFT.Open MSFT.High MSFT.Low MSFT.Close MSFT.Volume
## 2016-11-09     60.00     60.59    59.20      60.17    49632479
## 2016-11-10     60.48     60.49    57.63      58.70    57822394
## 2016-11-11     58.23     59.12    58.01      59.02    38767843
## 2016-11-14     59.02     59.08    57.28      58.12    40861850
## 2016-11-16     58.94     59.66       NA      59.65    26849497
## 2016-11-18     60.78     61.14    60.30      60.35    27686311
```

We can specify a specific date range as follows:

```
getSymbols('MSFT', src = "google", from = "2015-01-01", to = "2015-12-31")
```

```
## [1] "MSFT"
```

```
head(MSFT)
```

```
##            MSFT.Open MSFT.High MSFT.Low MSFT.Close MSFT.Volume
## 2015-01-02     46.66     47.42    46.54      46.76    27913852
## 2015-01-05     46.37     46.73    46.25      46.32    39673865
## 2015-01-06     46.38     46.75    45.54      45.65    36447854
## 2015-01-07     45.98     46.46    45.49      46.23    29114061
## 2015-01-08     46.75     47.75    46.72      47.59    29645202
## 2015-01-09     47.61     47.82    46.90      47.19    23944181
```

```
tail(MSFT)
```

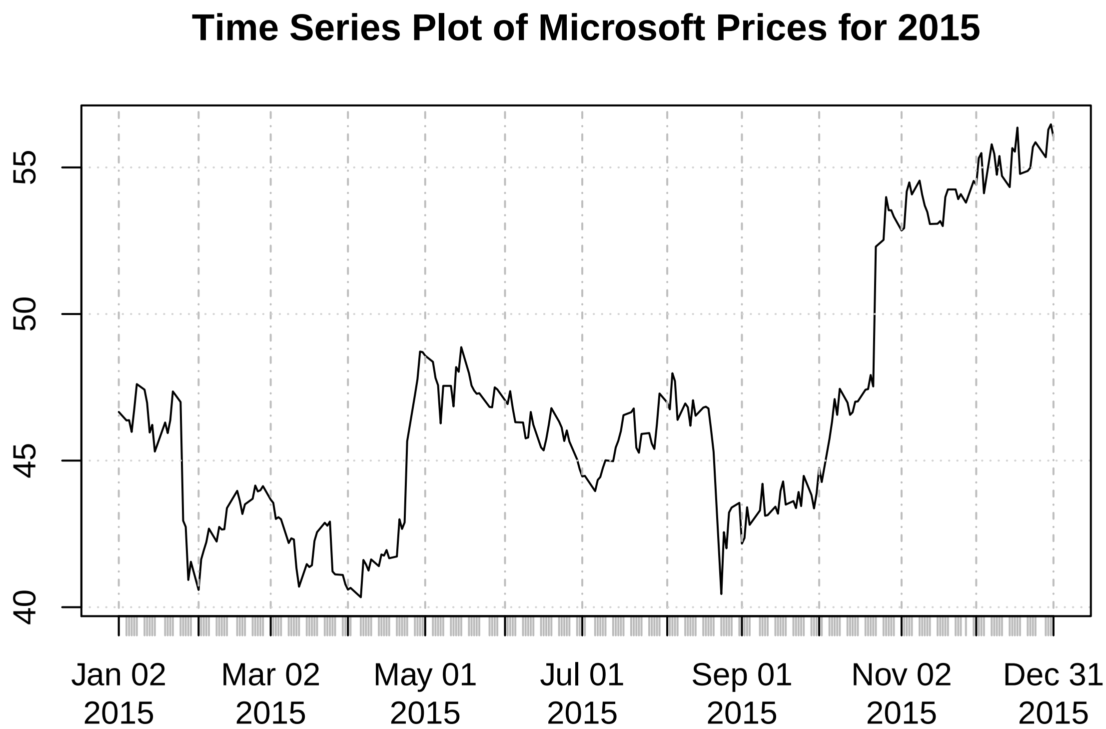
```
##            MSFT.Open MSFT.High MSFT.Low MSFT.Close MSFT.Volume
## 2015-12-23     55.70     55.88    55.44      55.82    27279832
## 2015-12-24     55.86     55.96    55.43      55.67     9570002
## 2015-12-28     55.35     55.95    54.98      55.95    22458293
## 2015-12-29     56.29     56.85    56.06      56.55    27731403
## 2015-12-30     56.47     56.78    56.29      56.31    21704505
## 2015-12-31     56.04     56.19    55.42      55.48    27334061
```

Making Plots

We can make very nice looking time-series plots with `xts` objects as follows:

```
plot(MSFT, main = "Time Series Plot of Microsoft Prices for 2015")
```

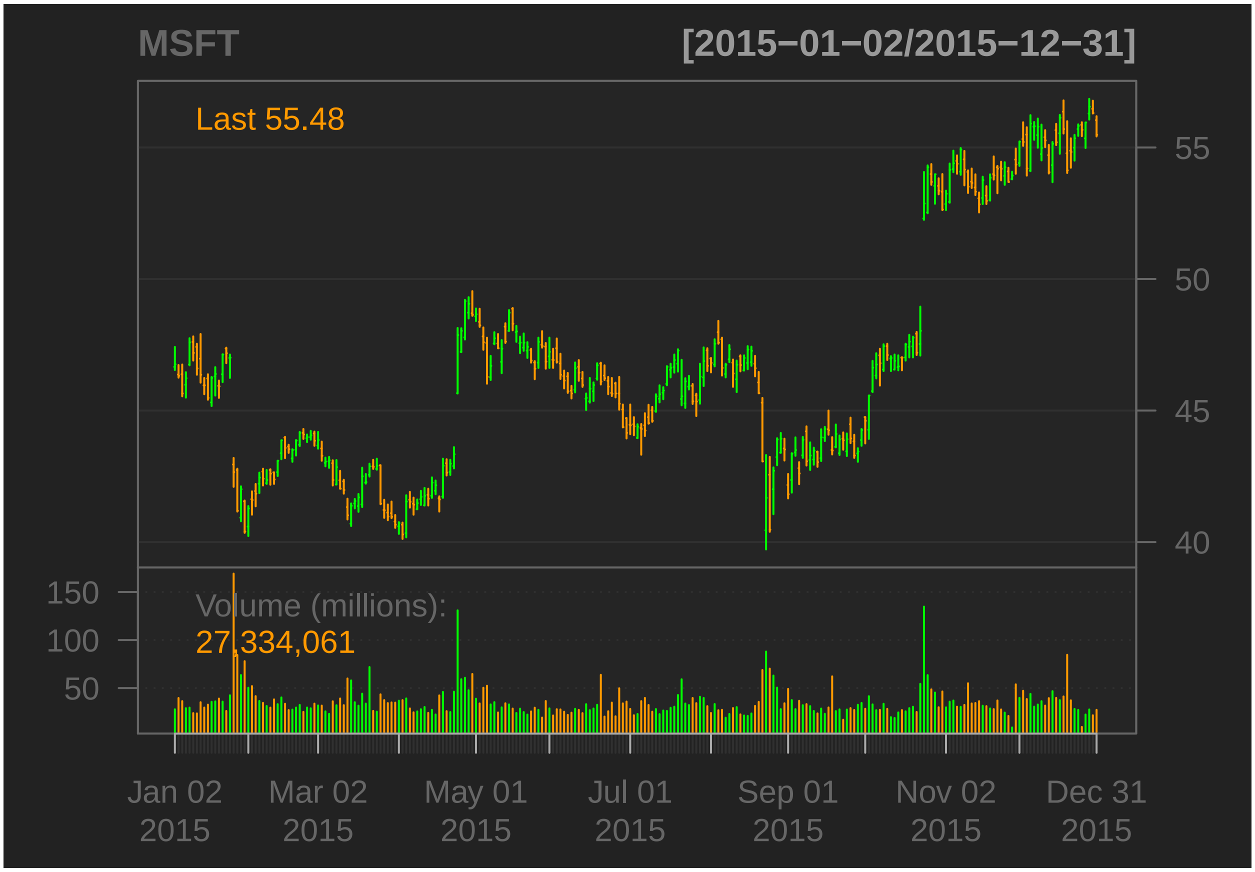
```
## Warning in plot.xts(MSFT, main = "Time Series Plot of Microsoft Prices for
## 2015"): only the univariate series will be plotted
```



There are additional parameters you can use to control more finely the output, so be sure to take a look at the help file for `plot.xts`.

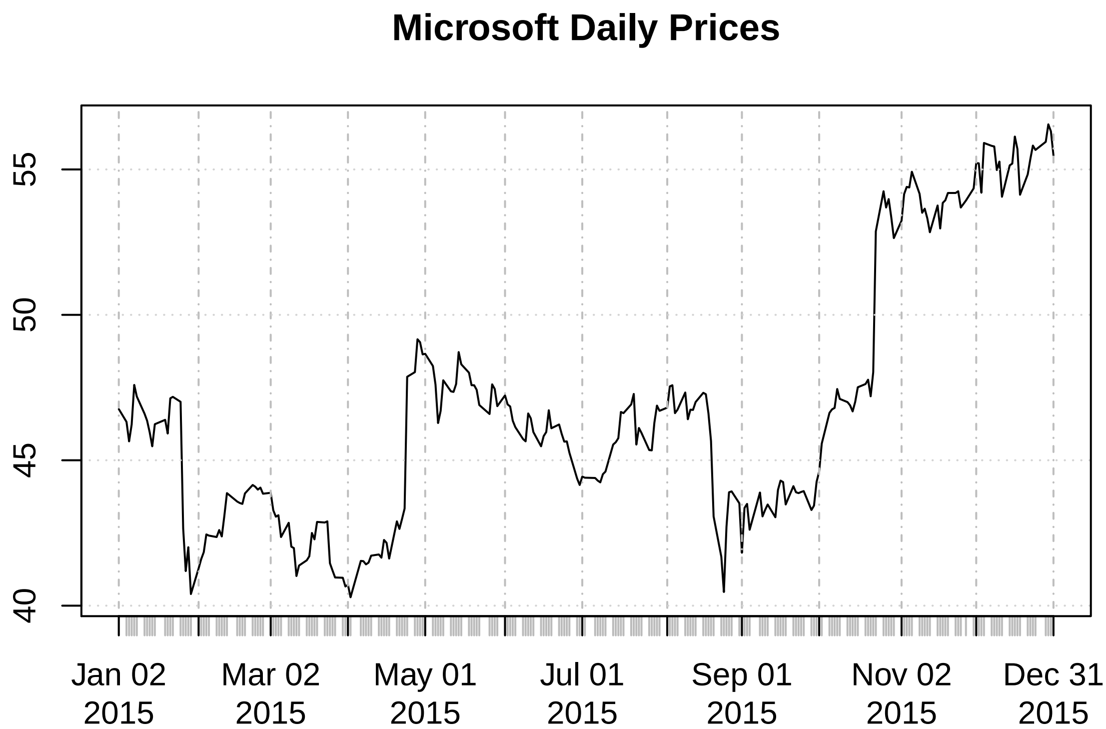
You can also do other neat plots, such as this one:

```
barChart(MSFT)
```



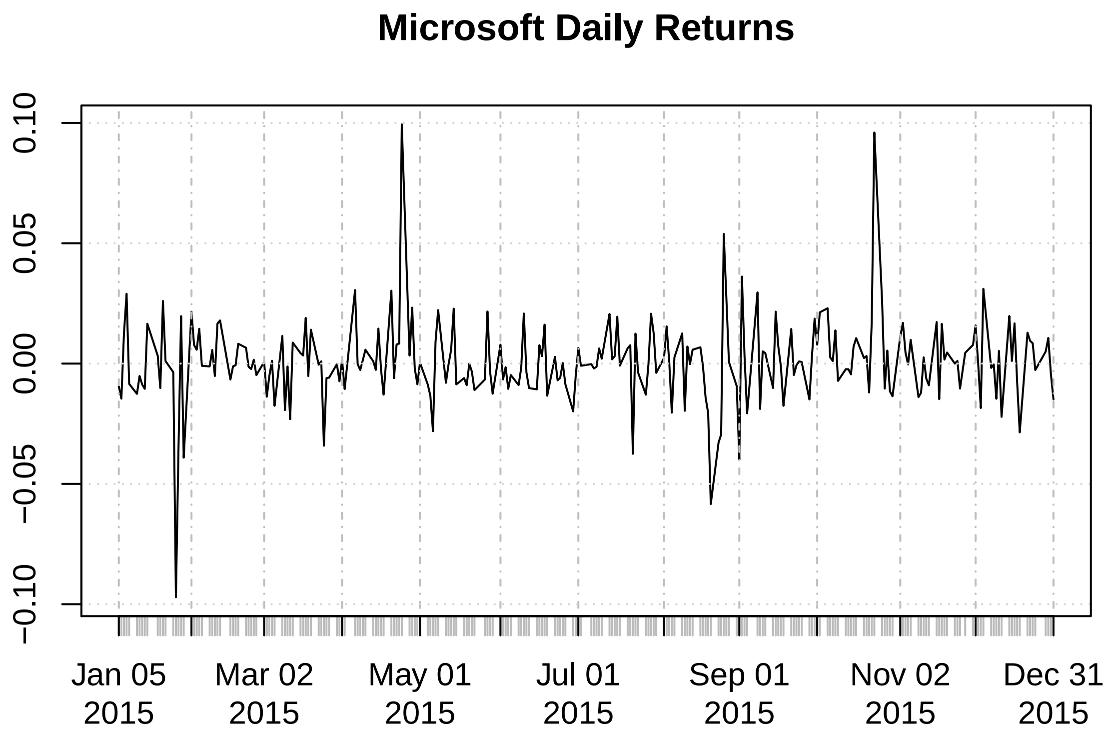
We can extract a single column as follows:

```
prc.close <- MSFT$MSFT.Close
plot(prc.close, main = "Microsoft Daily Prices")
```



We can also use some utility functions to transform prices into log returns as follows:

```
ret.close <- diff(prc.close, log=TRUE, na.pad=FALSE)
plot(ret.close, main = "Microsoft Daily Returns")
```



Fitting ARMA Models

First let's use the function `arima.sim` to simulate some data from a given ARMA process.

You can look at the help function for this function:

```
> help(arima.sim)
```

Let's simulate a $ARMA(2,2)$ process:

```
sim.data <- arima.sim(list(ar = c(0.4, 0.4), ma = c(0.6, -0.4)), n = 10000)
```

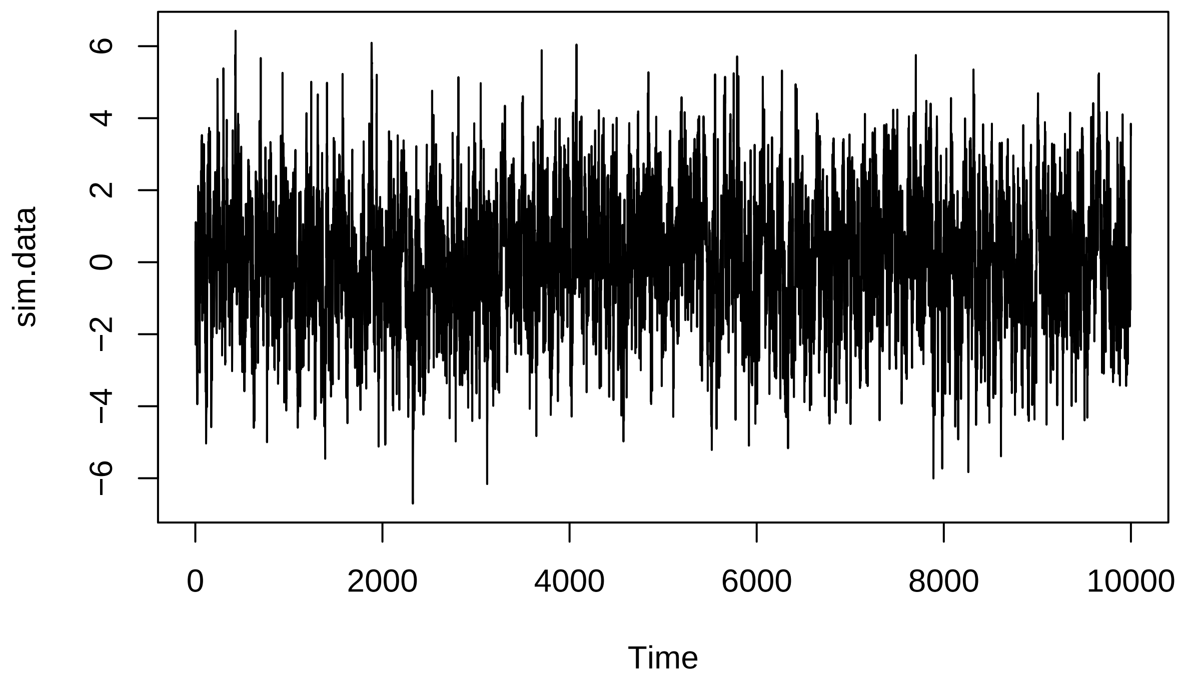
We can see what type of object this gives us:

```
data.class(sim.data)
```

```
## [1] "ts"
```

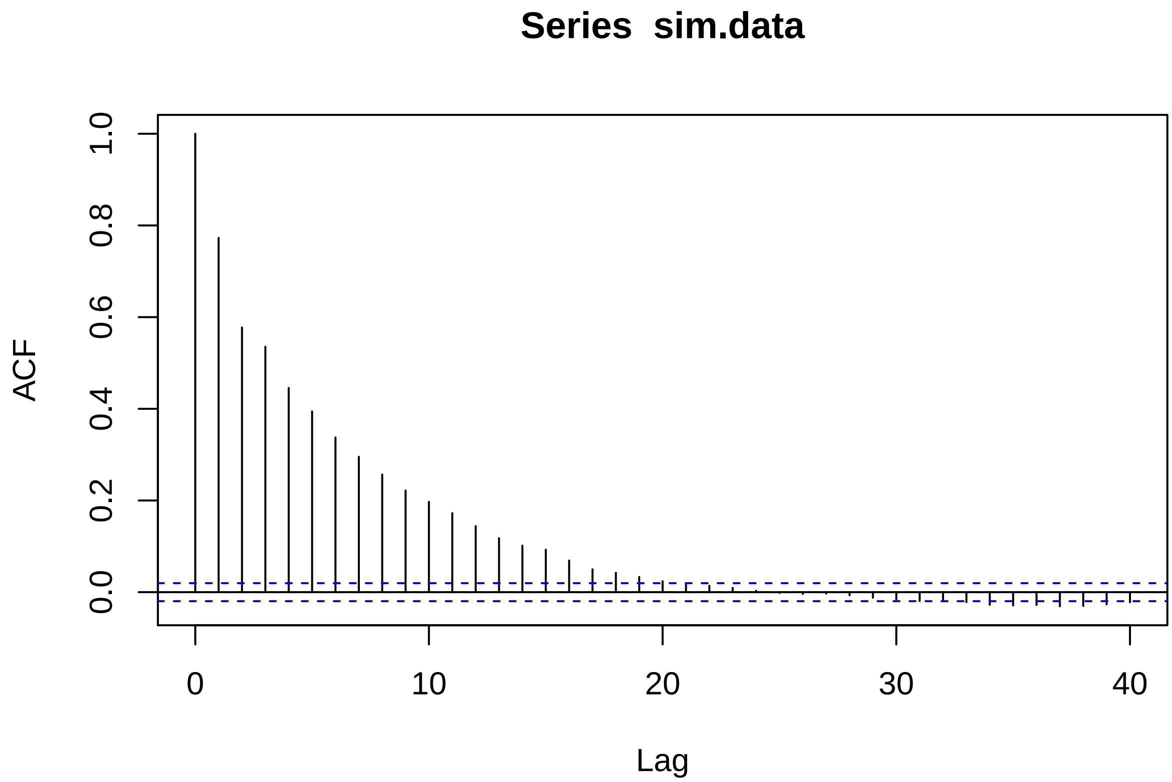
It's a `ts` object, which has it's own plot function:

```r
plot(sim.data)
```

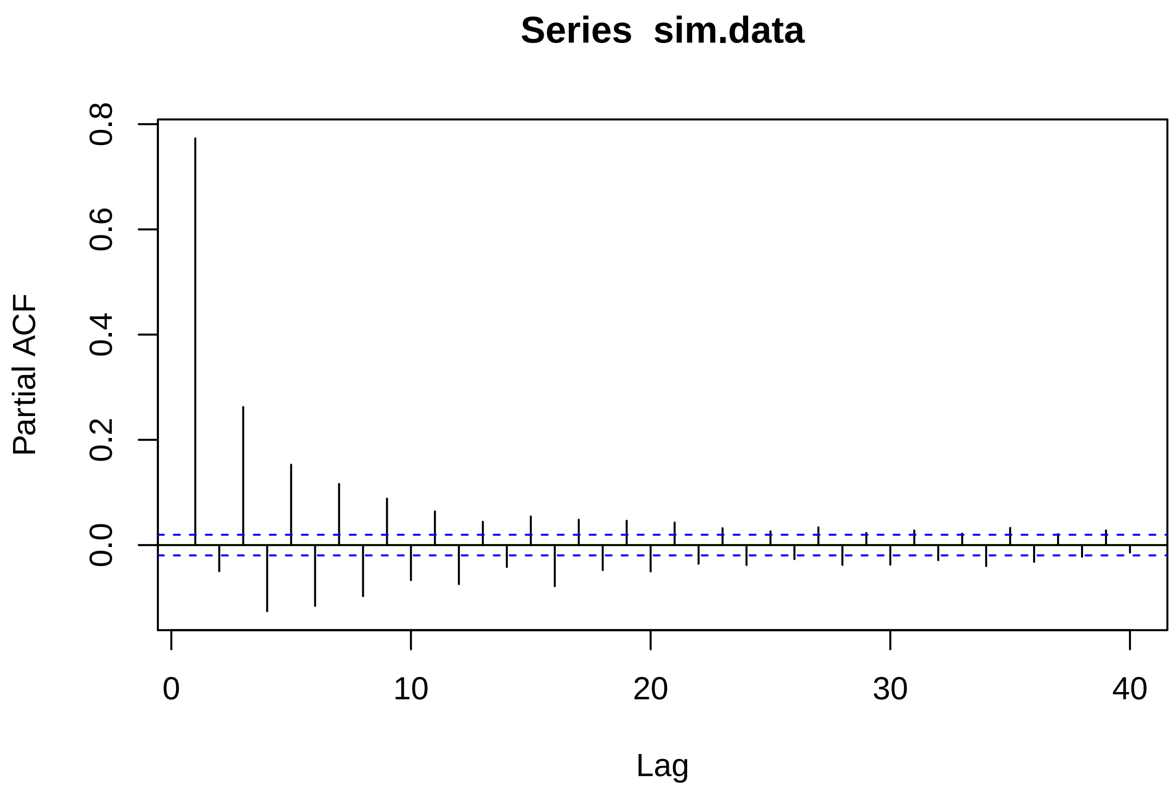


We can get the ACF and PACF plots as follows:

```r
acf(sim.data)
```




```
pacf(sim.data)
```



Let's fit the model knowing that $p = 2$ and $q = 2$:

```
fit <- arima(sim.data, order = c(2,0,2), method="ML")
```

We can retrieve the AIC and BIC:

```
aic <- round(AIC(fit), 3)
bic <- round(BIC(fit), 3)
print(paste("The AIC is:", aic, sep=" "))
```

```
## [1] "The AIC is: 28580.985"
```

```
print(paste("The BIC is:", bic, sep=" "))
```

```
## [1] "The BIC is: 28624.247"
```

Unfortunately, model specification requires a mixture of graphical analysis and these information criteria. For a helpful function please the following by Rob Hyndman (which utilizes his `forecast` package): https://www.otexts.org/fpp/8/7.

Let's try his `auto.arima` method of model building below for our simulated data:

```
require(forecast)
```

```
## Loading required package: forecast

## Loading required package: timeDate

## This is forecast 7.3
```

```
auto.arima(sim.data, max.p = 6, max.q = 6, stationary = TRUE, parallel=TRUE, num.cores = 8, stepwise=FA
```

```
## Series: sim.data
## ARIMA(1,0,2) with zero mean
##
## Coefficients:
##          ar1     ma1      ma2
##       0.9138  0.1796  -0.6234
## s.e.  0.0061  0.0133   0.0162
##
## sigma^2 estimated as 1.121:  log likelihood=-14759.55
## AIC=29527.1   AICc=29527.11   BIC=29555.95
```

Recall that ARMA processes are not necessarily unique, so don't be surprised if the procedure suggests a specification that differs from our known $ARMA(2,2)$ process. In this case it chose an $ARMA(1,2)$ model!

Note: also note that I set the `parallel` argument to `TRUE` and `num.cores` to 8. This allows for the model specification search to be done in parallel. Most modern CPUs have multiple cores. You may want to set these parameters differently for your machine. If you choose not to use parallel processing you should set `stepwise=TRUE`.

ARMA Modeling of Microsoft Returns

```
auto.arima(ret.close, max.p = 6, max.q = 6, stationary = TRUE, parallel = TRUE, num.cores = 8, stepwise
```

```
## Series: ret.close
## ARIMA(0,0,0) with zero mean
##
## sigma^2 estimated as 0.0003131:  log likelihood=656.5
## AIC=-1311   AICc=-1310.99   BIC=-1307.48
```

Notice that it selects an $ARMA(0,0)$ model, which essentially corresponds with the white noise process. This tends to fit with our theory from finance.