Time Series Analysis in R[*]

Jeric C. Briones

Department of Mathematics

Ateneo de Manila University

1 Preliminaries

We begin by doing preliminary analysis on simulated time series data. Codes used are in `0TSA.R`, `1LinearModels.R`, `2VolatilityModels.R`, and `3MTS.R`.

1.1 Loading Simulated Data

We begin by loading the simulated dataset `ar2.s` and `ma1.1.s` from the library `TSA`. This simulated time series data will be used for the succeeding preliminary analyses.

Aside from the loaded simulated data, we also generate another time series data using random standard normal deviates. Standard normal deviates are generated using `rnorm()`, where the required input is the number of deviates to generate. When unspecified, the default mean and standard deviation are `mean=0` and `sd=1`, respectively. The deviates are converted to time series data using the function `ts()`.

```r
# Time-series library
library("TSA")
library("tseries")

# simulated data
data("ar2.s")
data("ma1.1.s")
z = ts(rnorm(120))
```

By default, the parameters of `ts()` are `start=1`, `end=T`, and `frequency=1`. Here, `T` refers to the length of the time series data, while `frequency` refers to the number of observations in a given time step. Commonly used frequencies include `frequency=12` to represent monthly data, and `frequency=4` to represent quarterly data. These frequencies can easily by changed for a given time series object, as shown below.

```r
# change frequency; even if frequency was changed, index is unchanged
## set frequency to 10 per unit time
z1 = ts(z, frequency=10)
## set frequency to monthly, from Dec 2012 to Mar 2023
z2 = ts(z, start=c(2012,12), end=c(2023,3),frequency=12)
plot(z)
plot(z1)
plot(z2)
```

[*]Supplementary notes for MATH 62.2 Time Series and Forecasting, and MATH 271.2 Advanced Statistical Methods. Last updated Intersession, SY 2024-2025

Here, z1 has the same values as z. However, instead of having 120 unit time steps like the latter, the former has $\frac{120}{10} = 12$ unit time steps, where each unit time steps has 10 sub-steps. Similarly, z2 also has the same values as z but with monthly frequency (frequency=12). To indicate that the time series starts December 2012 and ends March 2023, the parameters start=c(2012,12) and end=c(2023,3) are used. It can be noticed that when z, z1, and z2 are plotted, the figures are similar except for the x-axis scale.

1.2 Exploratory Analysis

After loading the dataset, we begin performing exploratory tests to the data: test for autocorrelation and unit-root test. Test for autocorrelation is performed using Box.test(). To specify the portmanteau test used, we use the parameter type. Here, we consider the Ljung-Box Test, specified using type="Ljung". Moreover, by default, this function tests for lag-1 autocorrelation. To test for lag-l autocorrelation, the parameter lag=l is used. Recall that should there be no autocorrelation, modeling using linear time series model would not be meaningful[1].

```r
# check for autocorrelation
# default is lag = 1
Box.test(ar2.s, type="Ljung")
Box.test(ma1.1.s, type="Ljung")
Box.test(z, type="Ljung", lag=3) # test for lag-3 correlation
```

To test for stationarity, we perform unit-root test using augmented Dickey-Fuller Test[2] (adf.test()). Here, the null hypothesis is the presence of unit root, which means the time series data is not stationary. Thus, if the null hypothesis is rejected, the time series data is considered as stationary. Recall that should the data be non-stationary, preprocessing steps (such as differencing) needs to be performed.

```r
# test for stationarity (AR)
adf.test(ar2.s)
adf.test(ma1.1.s)
adf.test(z)
```

1.3 Model Building

After checking for autocorrelation and stationarity, we then proceed with time series modelling. We start with lag order selection. This is done by checking the ACF (using acf()) and PACF (using acf() and adding the parameter type="partial"). Alternatively, pacf() can also be used to check PACF.

```r
# compute for ACF
acf(ar2.s)
acf(ma1.1.s)
acf(z)

# compute for PACF
acf(ar2.s, type="partial")
acf(ma1.1.s, type="partial")
acf(z, type="partial")
```

[1] For illustrative purposes, we continue modelling z despite having no autocorrelation.

[2] Recall that MA processes are weakly stationary. So, we basically test for stationarity for AR processes. On the other hand, recall that the ADF test assumes an AR model.

Here, notice that based on their respective plots, there is no autocorrelation between the data and its lag-1 value, consistent with earlier results. This is because none of the ACF and PACF values of `z` are statistically significant. That is, since none of these values exceed the dotted lines, then none of them are statistically different from zero. On the other hand, for `ar2.s`, ACF values decay while PACF values cut off at lag $h = 2$ (with another significant ACF value at lag $h = 9$). For `ma1.1.s`, ACF values seem to cut off at lag $h = 1$ (but with other significant ACF value at lags $h = 5, 14$).

Based on these lags, we then estimate the model coefficients using `arima()`. The lag orders are specified using the parameter `order=c(p,0,q)`, where p refers to the AR lag order, while q the MA lag order[3]. Here, we test AR(2) and AR(9) for `ar2.s`, and MA(1), MA(5), and MA(14) for `ma1.1.s`. We also test AR(1) for `ar2.s` for illustrative purposes.

```r
# fit ARMA models
# order = AR lag, integration, MA lag
arima(ar2.s, order=c(1,0,0)) # test AR(1)
arima(ar2.s, order=c(2,0,0)) # test AR(2)
arima(ar2.s, order=c(9,0,0)) # test AR(9)

arima(ma1.1.s, order=c(0,0,1)) # test MA(1)
arima(ma1.1.s, order=c(0,0,5)) # test MA(5)
arima(ma1.1.s, order=c(0,0,14)) # test MA(14)
```

Recall that we can use AIC to determine the best model, with the *best* model having the lowest AIC. Based on the results, the time series data `ar2.s` is an AR(2) process (AIC of 331.95 vs 451.16 for AR(1) and 339.58 for AR(9)), while `ma1.1.s` is an MA(1) process (AIC of 363.66 vs 372.58 for MA(5) and 339.58 MA(14)).

2 Fitting Linear Time Series Models

We continue by loading the datasets `SP` and `usd.hkd` from the library `tseries`, and the dataset `strikes` from the library `itsmr`. The first dataset is the quarterly S&P composite index, from Quarter 1 of 1936 to Quarter 4 of 1977, the second is the daily USD/HKD exchange rate from January 1, 2005 to March 7, 2006 (with the FOREX rates seen in `usd.hkd$hkrate`), while the third is the annual number of USA union strikes from 1951-1980.

2.1 Data: S&P Composite Index

As earlier discussed, after loading the time series data, we begin with the preliminary tests, specifically the test for stationarity using `adf.test()`.

```r
# Quarterly S&P Composite Index, 1936Q1 - 1977Q4.
data("SP")

# test for stationarity (AR)
adf.test(SP)
```

Here, we note that the null hypothesis of having a unit root (that is, the data is non-stationary) was not rejected. Hence, preprocessing steps are needed to transform the data into a stationary process. Possible preprocessing steps include log transformation,

[3]The middle parameter, `0`, refers to the order of integration d, where $d = 0$ for ARMA processes.

differencing, and log differencing. In this example, we focus on log differencing, since the data values are quite large. Moreover, we note that log transformation is generally used to handle large-valued data, while differencing is mostly used to remove non-stationarity.

```r
# perform pre-processing
SP.ln = log(SP) # take log transformation
SP.ln.d = diff(SP.ln) # take first difference of log
plot(SP.ln.d)

# perform stationarity test for differenced data
adf.test(SP.ln.d)

# check autocorrelation
Box.test(SP.ln.d, type="Ljung")
```

Performing the unit-root test on the transformed data, we note that the resulting time series data is now stationary. Given it is stationary, we then proceed with the test for autocorrelation. Using the Ljung-Box Test, we note that the data has no autocorrelation since the test failed to reject the null hypothesis. As such, further modelling for this time series data will not be pursued.

2.2 Data: USD/HKD FOREX Rate

We begin by converting the current data into a time series object using the function `ts()`. Then, like earlier, we start with the preliminary tests `adf.test()` and `Box.test()`.

```r
# Daily USD/HKD exchange rate from January 1, 2005 to March 7, 2006
data("usd.hkd")
forex = ts(usd.hkd$hkrate)

# perform initial tests
adf.test(forex)
Box.test(forex, type="Ljung")
```

Here, both tests rejected their respective null hypotheses. Since the data is stationary and serially correlated, we then proceed with (lag) order determination.

```r
# check ACF, PACF
acf(forex)
acf(forex, type="partial")
```

Based on the ACF and PACF plots, notice that the ACF plot seems to cut off at lag $h = 1$, while the PACF plot seems to cut off at lag $h = 2$. Thus, we test for AR(2), MA(1), and ARMA$(2, 1)$.

```r
# check models, lower AIC is better
arima(forex, order=c(0,0,1)) # test MA(1)
arima(forex, order=c(2,0,0)) # test AR(2)
arima(forex, order=c(2,0,1)) # test ARMA(2,1)
```

Checking the AIC, the *best* model is ARMA$(2, 1)$ with an AIC of -1890.64 (vs -1888.41 for MA(1) and -1889.49 for AR(2)). Thus, `usd.hkd$hkrate` follows an ARMA$(2, 1)$ process.

2.2.1 Parameter Estimates

Given that the data follows an ARMA$(2, 1)$, then its functional form is given by

$$X_t = \phi_0 + \phi_1 X_{t-1} + \phi_2 X_{t-2} + Z_t + \theta_1 Z_{t-1},$$

where $\{X_t\}$ is `usd.hkd$hkrate`.

Parameter estimates are included as outputs of `arima()`. To be specific, estimation is done using the maximum likelihood approach, with the estimates for the coefficients stored in `arma21$coef`. Note that the `intercept` here is not $\hat{\phi}_0$. Rather it is $\hat{\mu}$, the estimate for μ. Using the fact that $\mu = \dfrac{\phi_0}{1-\phi_1-\phi_2}$, we get $\phi_0 = \mu(1-\phi_1-\phi_2)$. As such, the functional form of the model is

$$X_t = -0.0004 + 0.2899X_{t-1} - 0.0588X_{t-2} + Z_t - 0.4763Z_{t-1}.$$

While the results yielded $\hat{\mu} = -0.0005$, it can be noticed that the confidence interval[4] for $\hat{\mu}$ is $(-0.0014, 0.0004)$. Since the interval contains 0, it can be argued that $\hat{\mu}$ is not significantly different from 0.

2.2.2 Forecasting

Forecasting for ARMA(p,q) is done using `predict()`, where the first parameter is the `arima()` object (in this case, `arma21`), while the second parameter is the number of steps ahead `n.ahead=m`. The forecast values are stored in `arma21.p$pred`, while the associated standard errors[5] are in `arma21.p$se`. Values are arranged $\left\{X_{n+1}^{(n)}, X_{n+2}^{(n)}, \ldots, X_{n+m}^{(n)}\right\}$.

```r
library(forecast)
# from previous result, use ARMA(2,1)
arma21 = arima(forex, order=c(2,0,1))
arma21$coef
# forecast 5-step ahead
arma21.p = predict(arma21, n.ahead=5)

# alternative function for estimation and forecasting
# forecast() only works when Arima() is used
arma21.2 = Arima(forex, order=c(2,0,1))
arma21.2$coef
arma21.p2 = forecast(arma21.2, h=5)
```

Alternatively, the function `Arima()` from the library `forecast` can be used for estimation and forecasting. While both function similarly, the results of `Arima()` is compatible with the function `forecast()`. Compared to `predict()` which only provides the point estimate and standard error, `forecast()` also includes prediction intervals as its output, as well as plot compatibility. For this function, the parameter `n.ahead=m` is replaced with `h=m`. Moreover, forecast values are stored in `arma21.p2$mean`. Additionally, compared to `arima()` which only outputs the value of the information criterion AIC, `Arima()` provides the values for the information criteria AIC, BIC, and AICc as well.

2.2.3 Analyzing Residuals

After fitting a time series model, the last step is to check the residuals. Recall that one of the assumptions of time series models discussed is that the residuals are uncorrelated. To

[4]Here, *confidence interval* is approximated using $\hat{\beta} \pm e$, where e is the associated standard error for the estimate $\hat{\beta}$. For a 95% confidence interval, use $\hat{\beta} \pm 1.96e$.

[5]Here, the standard error is the square root of the prediction error. That is, $\sqrt{P_{n+m}^{(n)}}$.

check this assumption, we can use the functions `tsdiag()` and `Box.test()`, among other tests. The `tsdiag()` provides residual-related plots, including standardized residual plots, ACF plots, and the Ljung-Box test p-value plots[6]. On the other hand, the Ljung-Box test can also be performed again, but using the residuals (stored in `arma21$residuals`, or retrieved using the function `residuals()`) as the input this time.

Depending on the data domain, normality of residuals may also be required. For this, `qqnorm()` and `jarque.bera.test()` may be used. Normality of residuals can be checked visually using `qqnorm`. Moreover, the Jarque-Bera test can be used to check if the residuals are normal by checking for its kurtosis and skewness.

```
# residual diagnostics
# Ljung-Box have issues with degrees of freedom
# jarque.bera.test() is used to check for skewness/kurtosis
tsdiag(arma21)
jarque.bera.test(residuals(arma21))
```

Based on the results, it was observed that while the residuals were uncorrelated, they are not normally distributed. Since the only lag that exceeded the bands in the ACF plot is lag 0, then the residuals are uncorrelated. Moreover, since the null hypothesis that the data are from normal distribution was rejected for the Jarque-Bera test, then the residuals are white noise (but not Gaussian white noise).

2.3 Data: Strikes

Similar to previous datasets, we perform preprocessing steps and preliminary tests.

```
library(itsmr)
# annual number of USA union strikes, 1951-1980
# perform initial tests and transformations
strikes.l = log(ts(strikes))
strikes.l.d = diff(strikes.l)
strikes.l.d2 = diff(strikes.l.d)
Box.test(strikes.l.d2, type="Ljung-Box")
adf.test(strikes.l.d2)

# determine order using ACF/PACF
acf(strikes.l.d2)
pacf(strikes.l.d2)
```

Note that the data `strikes` had to be differenced twice to make it stationary. Based on the ACF and PACF plots, candidate models include AR(2), MA(1), and ARMA(1, 1). The model with the best AIC is the AR(2) model.

Note that the middle parameter in `order=c(p,d,q)` refers to the order of integration d. As such, depending on d and the input data, there are several ways to use `Arima()`. If $d = 0$ (that is, using `order=c(2,0,0)`), then the input data must be the **differenced** data (which is `strikes.l.d2`). On the other hand, if $d > 0$ (that is, using `order=c(2,2,0)`), then the input data must be the original data (which is `strikes.l`). It should also be noted that in the absence of differencing (that is, if $d = 0$), `Arima()` assumes non-zero mean and provides an estimate for it (as seen in `mean`[7]). On the other

[6]Some textbooks argue that the Ljung-Box test results in the function are problematic since there are issues with the degrees of freedom used in the test.

[7]For `arima()`, this is the `intercept`.

hand, if ARIMA(p, d, q) is fitted, with $d > 0$, then `Arima()` assumes zero mean, and `mean` is omitted in the estimation. As such, there will be differences in the coefficient estimates depending on the choice of `order=c(p,d,q)`. If estimated mean $\hat{\mu}$ is statistically zero (based on the resulting prediction interval), then the estimated coefficients should not differ that much. If we wish to exclude the mean from estimation, the parameter `include.mean=F` should be used.

```
# estimate coefficients
# methods: ARMA, ARIMA, Yule-Walker
# note that ARMA estimates a mean, while ARIMA assumes zero-mean
ar2 = Arima(strikes.l.d2, order=c(2,0,0))
ari2 = Arima(strikes.l, order=c(2,2,0))
ar2.nomean = Arima(strikes.l.d2, order=c(2,0,0), include.mean=F)
ar2.yw = ar.yw(strikes.l.d2, order=2)
```

Based from the results, it can be seen that estimates for ϕ_1 and ϕ_2 are close for the three approaches considered (-1.1397 vs -1.1401 vs -1.1401 for ϕ_1, -0.7796 vs -0.7800 vs -0.7800 for ϕ_2). Here, the first estimate used $d = 0$, the second $d = 2$, while the third $d = 0$ but with `include.mean=F`. On the other hand, if we are interested in estimation using Yule-Walker equations, the function `ar.yw()`, with AR(2) specified using the parameter `order=2`. Note that estimates using Yule-Walker equations may be different those obtained using MLE approach.

In functional form, if X_t is the number of strikes, $Y_t = \nabla^2 X_t$ the twice-differenced data, then using the results of `ari2$coef`,

$$\begin{aligned}
Y_t &= -1.1401Y_{t-1} - 0.7800Y_{t-2} + Z_t \\
Y_t + 1.1401Y_{t-1} + 0.7800Y_{t-2} &= Z_t \\
(1 + 1.1401\mathrm{B} + 0.7800\mathrm{B}^2)Y_t &= Z_t \\
(1 + 1.1401\mathrm{B} + 0.7800\mathrm{B}^2)(1 - \mathrm{B})^2 X_t &= Z_t.
\end{aligned}$$

As mentioned earlier, using `forecast()` allows us to plot the point estimate and prediction intervals for the forecast. It should be noted that the resulting forecast and its plot follows the input data. For example, the forecast for `ar2` (given by `ar2.p`) is for the twice differenced data. If the original log-transformed forecast is desired, the forecast must be *un*differenced. On the other hand, since the input for `ari2` is the original log-transformed data, the results for its forecast would be for the log-transformed data.

```
# forecast 5-step ahead, differenced data
ar2.p = forecast(ar2, h=5)
plot(ar2.p)

# forecast 5-step ahead, original data
ar2.p2 = forecast(ari2, h=5)
plot(ar2.p2)

# residual diagnostics
tsdiag(ar2)
jarque.bera.test(residuals(ar2))
```

For the forecast plot, note that the original data is in black, the forecast values are in blue, 80% prediction intervals in grey shade, and 95% prediction intervals in bluish grey shade. Moreover, residual diagnostics are again performed. Here, the residuals are uncorrelated (since only ρ_0 is significant) and normally distributed (since p-value was large).

3 Fitting Seasonal Models

We continue by loading the built-in dataset `AirPassengers`. This dataset is the monthly airline passenger numbers from 1949-1960. That is, the dataset has a frequency of 12.

3.1 Exploratory Analysis

We begin by performing preprocessing steps and preliminary tests.

```r
# Monthly Airline Passenger Numbers 1949-1960
air = AirPassengers

# perform initial transformations
air.l = log(air)
air.l.d = diff(air.l)
plot(cbind(air, air.l,air.l.d))

# perform initial tests
Box.test(air.l.d,type="Ljung")
adf.test(air.l.d)
```

Based on the plots, it is clear that the data has trend and seasonality. However, results from `adf.test()` suggest that the data is already stationary. This result highlights how the ADF test is not as effective for detecting non-stationarity caused by seasonality.

Seasonal differencing can be done using `diff()` by setting the second parameter as the seasonal integration order D. For example, if we are interested with $\nabla_{12}X_t$ for seasonal differencing, then we code this as `diff(Xt, 12)`.

```r
# do seasonal differencing
# use default frequency as s
air.l.d.12 = diff(air.l.d,12)
plot(cbind(air.l,air.l.d, air.l.d.12))

# determine order
# note that lag 1 = step 12
acf(air.l.d)
acf(air.l.d, lag=100)
pacf(air.l.d, lag=100)
```

Using the parameter `lag=100` in `acf()` and `pacf()` allows us to see more lags, which is useful when determining lag order for seasonal data. Based on the ACF and PACF plots, $\text{sAR}(1)_{12}$ is a candidate model. This is because the PACF plot cuts off after lag $Ps = 12$, and the ACF plot tails off at lags $12k$. However, notice that there are non-zero values for lags that are not of the form $12k$. As such, mixed seasonal ARIMA model should be considered. For simplicity, the non-seasonal components for the model considered would be AR(1), MA(1), and ARMA$(1, 1)$ only.

3.2 Model Building

In estimating the coefficients, instead of using `air.l.d` as input, we use `air.l` and set $d = 1$. That is, the parameters `order=c(1,1,1)`, `order=c(0,1,1)`, and `order=c(1,1,0)` are used for the non-seasonal component of `Arima()`. On the other hand, the seasonal component is indicated by the parameter `seasonal=list(order=c(P,D,Q), period=s)`. If `period=s` is dropped, the frequency of the `ts()` object is used by default. In this case, the seasonal component parameter becomes `seasonal=order(P,D,Q)`. For this dataset, sAR$(1)_{12}$ is coded as `seasonal=list(order=c(1,1,0), period=12)`. Based on the resulting AIC, sARIMA$(0,1,1) \times (1,1,0)_{12}$ is the *best* model (-475.4664 vs -477.4053 vs -474.8188).

```r
# fit using arima
sarima.ar1 = Arima(air.l, order=c(1,1,1), seasonal=list(order=c(1,1,0),
    period=12))
sarima.ma1 = Arima(air.l, order=c(0,1,1), seasonal=list(order=c(1,1,0),
    period=12))
sarima.arma11 = Arima(air.l, order=c(1,1,0), seasonal=list(order=c
  (1,1,0), period=12))

# selecting best model
c(sarima.ar1$aic, sarima.ma1$aic, sarima.arma11$aic)
```

As previously mentioned, the mixed seasonal model can be implemented in different ways. To provide different plots for the forecast, three different variations for coding sARIMA$(0,1,1) \times (1,1,0)_{12}$ are considered:

- `sarima.model` uses `air.l` as input. As such $d = 1$ and $D = 1$. Since the input is the log-transformed data, the forecast `sfor` is log forecasts.
- `sarima.model.d` uses `air.l.d` as input. As such $d = 0$ and $D = 1$. Since the input is the differenced log-transformed data, the forecast `sfor2` is differenced log forecasts.
- `sarima.model.d.12` uses `air.l.d.12` as input. As such $d = 0$ and $D = 0$. Since the input is the season-differenced and first-differenced log-transformed data, the forecast `sfor3` also underwent the same transformation. To make the estimates comparable, `include.mean=F` was also used.

```r
# fit using Arima, the forecast
# if period is omitted, default is frequency
sarima.model = Arima(air.l,order=c(0,1,1),seasonal=list(order=c(1,1,0),
   period=12))
sfor = forecast(sarima.model, h=12)
plot(sfor)
sfor$mean

# input is differenced data
sarima.model.d = Arima(air.l.d,order=c(0,0,1),season=c(1,1,0))
sfor2 = forecast(sarima.model.d, h=12)
plot(sfor2)

# input is seasonal differenced data
sarima.model.d.12 = Arima(air.l.d.12,order=c(0,0,1),season=c(1,0,0),
   include.mean=F)
sfor3 = forecast(sarima.model.d.12, h=12)
plot(sfor3)
```

In functional form, if X_t is the number of airline passengers, and using the results of `sarima.model$coef`, we get

$$(1 + 0.4743\mathrm{B})(1 - \mathrm{B}^{12})(1 - \mathrm{B})X_t = (1 - 0.4423\mathrm{B})Z_t.$$

4 Fitting Volatility Models

We continue by revisiting the dataset `usd.hkd` from the library `tseries`.

4.1 Data: USD/HKD FOREX Rate

Recall that the data `usd.hkd$hkrate` follows an ARMA$(2,1)$ process, and that the best linear time series model for this dataset is `arma21`. We then begin by checking for auto-correlation for the residuals and squared residuals.

```r
# get the residuals after fitting ARMA(2,1)
res.arma21 = residuals(arma21)

# check the residuals
acf(res.arma21)
pacf(res.arma21)
Box.test(res.arma21, type="Ljung")

# check the squared residuals
acf(res.arma21^2)
pacf(res.arma21^2)
Box.test(res.arma21^2, type="Ljung")
```

The resulting correlograms clearly indicate that the residuals are uncorrelated, as there are no lags that exceed the bands. However, the square residuals are serially correlated since there are significant lags. Since lag $h = 1$ is significant, we continue with ARCH(1).

4.1.1 Two-Pass Estimation Method

We start with the two-pass estimation method where we model the mean and volatility equations separately. Note that we do not use the parameter `include.mean=F` since we want $\alpha_0 > 0$.

```r
# two-pass estimation
vol.ar1 = Arima(res.arma21^2, order=c(1,0,0))
vol.ar1$coef
```

Using this approach, we obtain

$$\begin{cases} Y_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= 0.0005 + 0.2517 Y_{t-1}^2 \end{cases},$$

where $\hat{\alpha}_0 := \hat{\phi}_0 = 0.0005 = \hat{\mu}(1 - \hat{\phi}_1)$. A quick check with the obtained standard errors shows that the approximate for the confidence intervals for $\hat{\alpha}_0$ and $\hat{\alpha}_1$ do not contain zero.

4.1.2 Initial Joint Model Fitting

We estimate the model coefficients using `garchFit()` from the library `fGarch`. The volatility model is specified using `~garch(m,s)`, where m refers to the order for Y_t^2 while s the order for σ_t^2. In the absence of a mean equation, ARMA$(0,0)$ is assumed. That is, $X_t = \mu + Y_t$. If the mean equation ARMA(p,q) is known, the parameter becomes `~arma(p,q)+garch(m,s)`.

```r
# Time-series library
library(fGarch)

# fit ARMA(p,q)-ARCH(1) model
vol = garchFit(~garch(1,0),data=forex) #arma(0,0)
arma21.vol = garchFit(~arma(2,1)+garch(1,0),data=forex) #arma(2,1)
summary(vol)
summary(arma21.vol)
```

The model vol gives

$$\begin{cases} X_t &= -0.0012 + Y_t \\ Y_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= 0.0005 + 0.4989 Y_{t-1}^2 \end{cases},$$

while the model arma21.vol yields

$$\begin{cases} X_t &= -0.0004 + 0.3120 X_{t-1} - 0.0952 X_{t-2} + Y_t - 0.4908 Y_{t-1} \\ Y_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= 0.0004 + 0.4491 Y_{t-1}^2 \end{cases},$$

where $\hat{\phi}_0 = -0.0004 = \hat{\mu}(1 - \hat{\phi}_1 - \hat{\phi}_2)$. Notice that while the estimated coefficients for the mean equation obtained using the two-pass and the joint estimation methods are different, it can be argued that they are statistically similar based on their resulting standard errors and confidence intervals[8]. Moreover, note that for the first model, $\hat{\mu}$ was not significant, while $\hat{\alpha}_0$ (labelled omega in the output) and $\hat{\alpha}_1$ are. On the other hand, for the second model, $\hat{\mu}$, $\hat{\phi}_1$, and $\hat{\phi}_2$ were not significant, while $\hat{\theta}_1$, $\hat{\alpha}_0$, and $\hat{\alpha}_1$ are. However, it should be noted that $\hat{\phi}_1$ was significant at the level $\alpha = 0.1$. As such, a possible reduced model to check is $\text{ARMA}(1,1) + \text{ARCH}(1)$.

4.1.3 Refining the Model

Using the reduced model, we fit the $\text{ARMA}(1,1) + \text{ARCH}(1)$ into the data.

```r
# refine model
arma11.vol = garchFit(~arma(1,1)+garch(1,0),data=forex)
summary(arma11.vol)
arma11.vol@fit$coef
arma11.vol@fit$se.coef
```

The point estimates of the coefficients and their corresponding standard errors are stored in arma11.vol@fit$coef and arma11.vol@fit$se.coef, respectively. In functional form, the $\text{ARMA}(1,1) + \text{ARCH}(1)$ model is given by

$$\begin{cases} X_t &= -0.0002 + 0.3429 X_{t-1} + Y_t - 0.5875 Y_{t-1} \\ Y_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= 0.0005 + 0.4331 Y_{t-1}^2 \end{cases},$$

with all the estimated coefficients except for $\hat{\mu}$ significant. Moreover, different information criteria for each model[9] can be seen below:

[8]Using $\hat{\phi}_1 \pm e$, the estimated C.I. for $\hat{\phi}_1$ in arma21 is $(0.0900, 0.4898)$, while in arma21.vol is $(0.1312, 0.4928)$. Since the intervals overlap, then the two estimates for $\hat{\phi}_1$ are arguably similar.

[9]These can be obtained using model@fit$ics. For example, the information criteria for arma11.vol are stored in arma11@fit$ics

Model	AIC	BIC	SIC
vol	-4.5239	$\mathbf{-4.4956}$	-4.5239
arma21.vol	$\mathbf{-4.5360}$	-4.4794	$\mathbf{-4.5364}$
arma11.vol	-4.5349	-4.4877	-4.5351

Depending on the choice of information criterion, the *best* model would vary. Interestingly, while arma21.vol had a smaller AIC, arma11.vol had a smaller BIC. For simplicity, we use arma11.vol as the best model.

4.1.4 Checking Standardized Residuals

After fitting the model, the last step is to check the model residuals. Recall that the resulting standardized residuals must be an iid sequence.

```r
# check standardized residuals
fit.vol = volatility(arma11.vol)
fit.vol.sr = residuals(arma11.vol) / volatility(arma11.vol)
plot(fit.vol, type="l")
plot(fit.vol.sr, type="l")

# check standardized residuals
acf(fit.vol.sr)
pacf(fit.vol.sr)
Box.test(fit.vol.sr, type="Ljung", lag=7)

# check standardized squared residuals
acf(fit.vol.sr^2)
pacf(fit.vol.sr^2)
Box.test(fit.vol.sr^2, type="Ljung")
```

From the correlogram results, except for lag $h = 7$, there seems to be no significant lags for the standardized residual fit.vol.sr. Doing the Ljung-Box test for lag $h = 7$, the null hypothesis that there are no serial correlation is not rejected, with p-value 0.1426. The same conclusion can be reached for the standardized squared residuals.

On the other hand, using the information from summary(garchFit()) regarding the different tests for the standardized residuals, it seems that these residuals are not normal, as supported by the results of the Jarque-Bera Test and Shapiro-Wilk Test[10]. This is because the p-value for these two tests are relatively zero regardless of the model used.

4.1.5 Forecasting

Forecasting for ARMA$(p, q)+$GARCH(m, s) is done using predict(), where the first parameter is the garchFit() object (in this case, arma11.vol), while the second parameter is the number of steps ahead n.ahead=m. An optional parameter, plot=T, can be included to plot the forecasts. The forecast values for $\{X_t\}$ are stored in forex.p$meanForecast, while the associated standard errors are in forex.p$meanError. On the other hand, forecast values for $\{\sigma_t\}$ are stored in forex.p$standardDeviation.

```r
# forecast values
forex.p = predict(arma11.vol, n.ahead=5, plot=T)
forex.p
```

[10]Recall that the first test checks for normality using skewness and kurtosis, while the second uses order statistics. In both cases, the null hypothesis is that the residuals are normal.

5 Fitting Vector ARMA Models

We continue by loading the datasets `cmort`, `tempr`, and `part` from the library `astsa`. These datasets are cardiovascular mortality, temperature, and particulate levels from the Los Angeles pollution study by Shumway et al. (1988) over the 10 year period 1970-1979.

```r
# Time-series library
library(astsa)
library(MTS)

# cardiovascular mortality (cmort), temperature (tempr), and
  particulate levels (part)
# from the LA pollution study over the 10 year period 1970-1979.
x = cbind(cmort, tempr, part)
```

5.1 Model Building

We estimate the model coefficients using `VARMA()` from the library `MTS`. The lag orders are specified using the parameters `p` and `q`, where p refers to the VAR lag order, while q the VMA lag order. Similar to the univariate case, if we wish to exclude the mean $\boldsymbol{\mu}$ from estimation, the parameter `include.mean=F` should be used.

```r
# Build VAR(1) model using MTS library
m.var1 = VARMA(x, p=1, q=0, include.mean=F)

# coefficients Phi (for AR) and Theta (for MA) are stored in Phi and
  Theta, respectively
# covariance matrix Sigma is in Sigma
m.var1$Phi
m.var1$Theta
m.var1$Sigma
```

The estimates for the model parameters are stored in `m.var1$Phi`, `m.var1$Theta`, and `m.var1$Sigma` for the VAR coefficient $\boldsymbol{\Phi}$, VMA coefficient $\boldsymbol{\Theta}$, and covariance matrix $\boldsymbol{\Sigma}$, respectively[11]. In functional form, the VAR(1) is given by

$$\begin{bmatrix} X_{M,t} \\ X_{T,t} \\ X_{P,t} \end{bmatrix} - \begin{bmatrix} 0.9563 & 0.0422 & 0.0109 \\ 0.2013 & 0.8878 & -0.2056 \\ 0.3169 & -0.0658 & 0.5049 \end{bmatrix} \begin{bmatrix} X_{M,t-1} \\ X_{T,t-1} \\ X_{P,t-1} \end{bmatrix} = \begin{bmatrix} \varepsilon_{M,t} \\ \varepsilon_{T,t} \\ \varepsilon_{P,t} \end{bmatrix},$$

where $X_{M,t}$, $X_{T,t}$, $X_{P,t}$ are the cardiovascular mortality, temperature, and particulate levels at time t.

5.2 Forecasting

Forecasting for VARMA(p, q) is done using `VARMApred()`, where the first parameter is the `VARMA()` object (in this case, `m.var1`), while the second parameter is the number of steps ahead `h=m`. The forecast values for $\{\boldsymbol{X}_t\}$ are stored in `m.var1.p$pred`, while the associated standard errors are in `m.var1.p$se.error`.

```r
# forecast values
m.var1.p = VARMApred(m.var1, h=5)
```

[11]For $p > 1$, the matrix stored in `m.var1$Phi` is the concatenated matrix $[\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2, \ldots, \boldsymbol{\Phi}_p]$. Similarly, for $q > 1$, `m.var1$Theta` is the concatenated matrix $[\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \ldots, \boldsymbol{\Theta}_q]$.