Tutorial: Forecasting

by Kevin Kotzé

1 The basic setup

To have a look at the first program for this session, open the project file `T4-forecast.Rproj`. The first script that we are going to work through is titled `T4_forecast.R`, which looks to generate a forecast for South African output growth. Once again, the first thing that we do is clear all variables from the current environment and close all the plots. This is performed with the following commands:

```
rm(list=ls())
graphics.off()
```

Thereafter, we will install the packages that we will use for this session, if you have not done so already. If you are using your personal machine then these would have been installed during the second tutorial, however, if you are using one of the University machines then you may need to install these packages with the following commands:

```
devtools::install_gitlab("KevinKotze/sarb2020q1")
devtools::install_github("KevinKotze/tsm")
install.packages('strucchange', repos='https://cran.rstudio.com/', dependencies=TRUE)
install.packages('forecast', repos='https://cran.rstudio.com/', dependencies=TRUE)
```

The next step is to make sure that you can access the routines within these packages by making use of the `library` command, which would need to be run regardless of the machine that you are using.

```
library(tsm)
library(forecast)
library(strucchange)
library(tidyverse)
library(lubridate)
library(sarb2020q1)
```

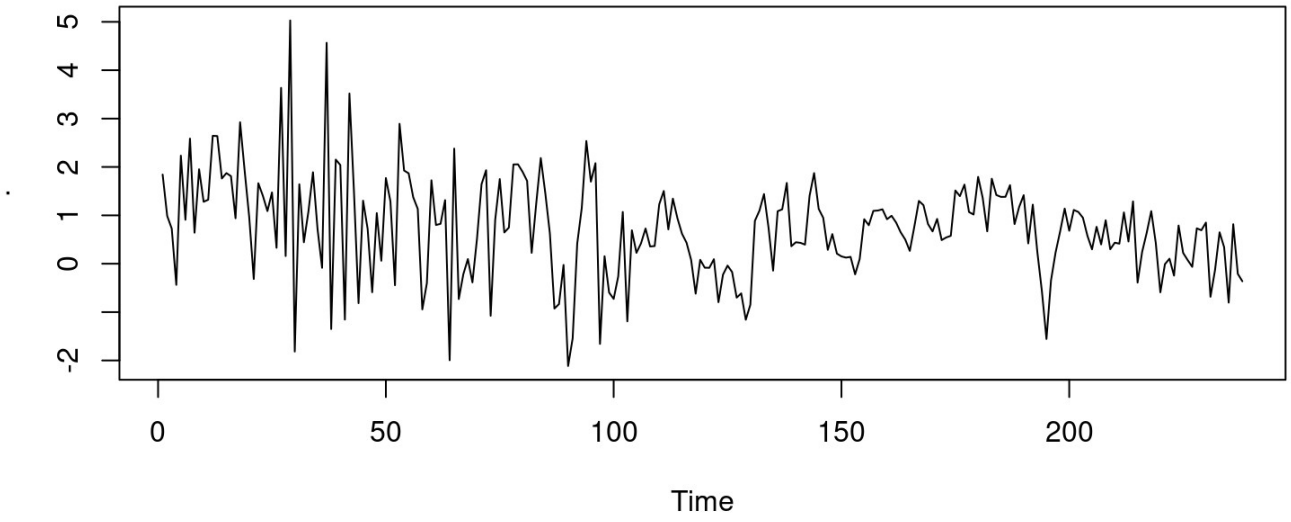
2 Transforming the data

As noted previously, South African GDP data is contained in the `sarb2020q1` package and it has the mnemonic `KBP6006D`. Hence, to retrieve the data from the package and create the tibble `gdp`, which includes a column for the growth rate and the lag of the growth rate, we execute the following commands:

```
gdp <- sarb_quarter %>%
  select(date, KBP6006D) %>%
  mutate(growth = 100 * ((KBP6006D / lag(KBP6006D)) - 1),
         grow_lag = lag(growth)) %>%
  drop_na()
```

To make sure that these calculations and extractions have been performed correctly, we inspect a plot of the data.

```
gdp %>%
  pull(growth) %>%
  plot.ts()
```

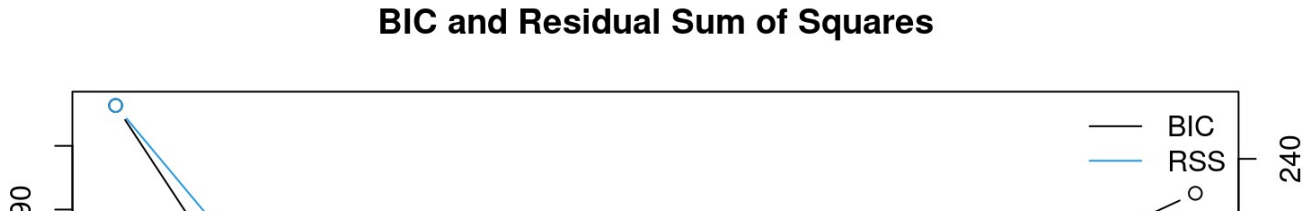


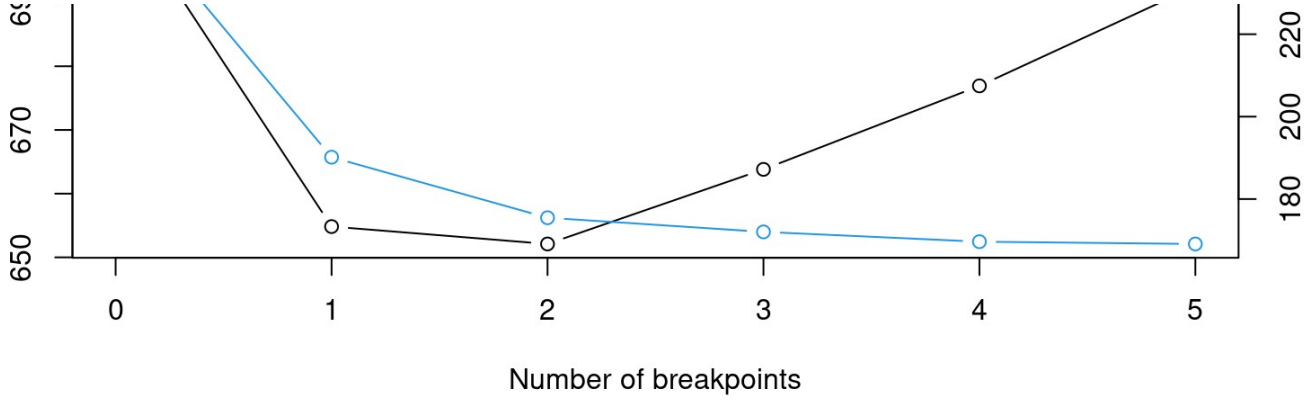
As we did before, we can proceed by checking for structural breaks. In this instance, we once again make use of an AR(1) model and the Bai and Perron (2003) statistic:

```
sa_bp <- breakpoints(growth ~ grow_lag, data = gdp, breaks = 5)
summary(sa_bp)
```

```
##
##   Optimal (m+1)-segment partition:
##
## Call:
## breakpoints.formula(formula = growth ~ grow_lag, breaks = 5,
##     data = gdp)
##
## Breakpoints at observation number:
##
## m = 1   42
## m = 2   42 78
## m = 3   42 78         203
## m = 4   42 78 130     203
## m = 5   42 78 130 166 203
##
## Corresponding to breakdates:
##
## m = 1   0.176470588235294
## m = 2   0.176470588235294 0.327731092436975
## m = 3   0.176470588235294 0.327731092436975
## m = 4   0.176470588235294 0.327731092436975
## m = 5   0.176470588235294 0.327731092436975
##
## m = 1
## m = 2
## m = 3
## m = 4   0.546218487394958
## m = 5   0.546218487394958 0.697478991596639
##
## m = 1
## m = 2
## m = 3   0.852941176470588
## m = 4   0.852941176470588
## m = 5   0.852941176470588
##
## Fit:
##
## m   0     1     2     3     4     5
## RSS 253.0 190.2 175.5 172.0 169.6 169.1
## BIC 706.3 654.8 652.1 663.8 676.9 692.6
```

```
plot(sa_bp, breaks = 5)
```






```
gdp %>%
  slice(sa_bp$breakpoint)
```

```
## # A tibble: 2 x 4
##   date       KBP6006D growth grow_lag
##   <date>        <dbl>  <dbl>    <dbl>
## 1 1970-10-01  1034117   3.52   -1.15
## 2 1979-10-01  1344927   2.05    0.746
```

After performing the exercise on successive subsamples of the data, the results suggest that one of the last significant structutral breaks arises towards the end of 1980.

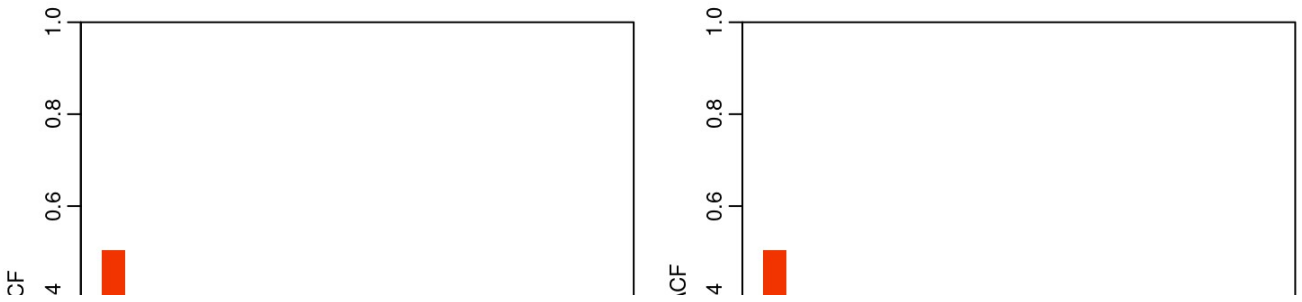
3 Estimating the model parameters

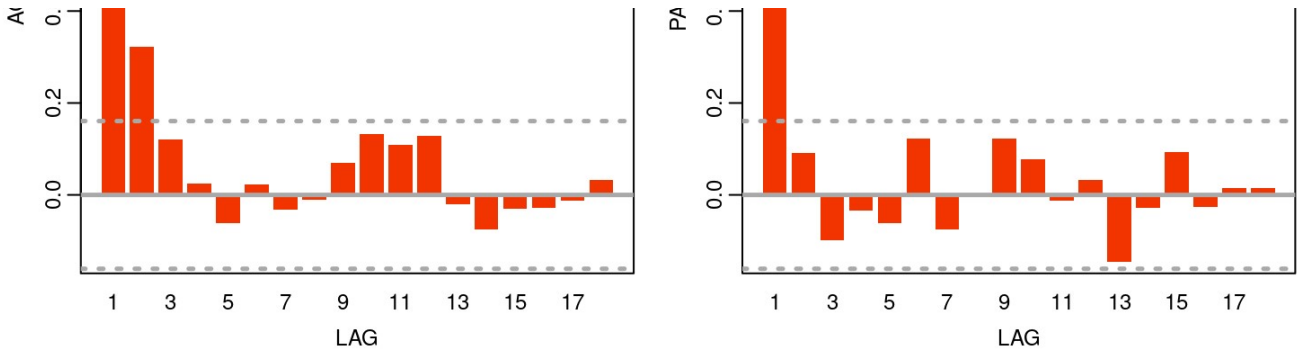
The subsample of the data that starts in 1981 is contained in the `y` object.

```
y <- gdp %>%
  filter(date > ymd('1981-01-01')) %>%
  select(date, growth)
```

To consider the degree of potential persistence we plot the autocorrelation and partial autocorrelation functions.

```
y %>%
  pull(growth) %>%
  ac()
```





This variable has a bit of persistence and is certainly stationary.

4 Recursive forecast generation and evaluation

Let us now proceed with a recursive forecasting exercise. What we would like to do is set up a few matrices for the actual data and forecasts, where each row pertains to a subsequent period of time and each column would refer to the additional step ahead forecast. In this exercise, we are going to make use of a forecasting horizon of eight-steps (i.e. two years) and an out-of-sample period of ten-years (i.e. forty observations). Using the notation that was provided in the notes, this would imply that:

```
H <- 8
P <- 40
R <- dim(y)[1] - (P + H) + 1
```

Should we want we can split the data into `train` and `test` subsamples. Such datasets could be created as follows:

```
train <- y %>%
  slice_head(n = R)

test <- y %>%
  slice_tail(n = (P + H - 1))
```

To inspect the last few observations in the initial training dataset along with the first few observations of the testing dataset:

```
train %>%
  tail()
```

```
## # A tibble: 6 x 2
##   date       growth
##   <date>      <dbl>
## 1 2006-10-01  1.38
## 2 2007-01-01  1.62
## 3 2007-04-01  0.820
## 4 2007-07-01  1.17
## 5 2007-10-01  1.42
## 6 2008-01-01  0.420
```

```
test %>%
  head()
```

```
## # A tibble: 6 x 2
##   date       growth
##   <date>      <dbl>
## 1 2008-04-01  1.22
## 2 2008-07-01  0.239
## 3 2008-10-01 -0.569
## 4 2009-01-01 -1.56
## 5 2009-04-01 -0.343
## 6 2009-07-01  0.232
```

To store the actual data that we will use to evaluate the forecast, we need a matrix that has the following dimensions.

```
actual <- matrix(data = rep(0, P * H), ncol = H)
```

This matrix would need to values that pertain to specific observations. For illustrative purposes, we can substitute the date into each cell with the following commands.

```
illust <- y %>%
  mutate(num = seq(1, n()))

for (i in 1:P) {
  first <- R + i
  last <- first + H - 1
  actual[i, 1:H] <- illust$num[first:last]
}
```

Note that as we move down or to the right, the number of the observation increases by one. Note also that the last observation number pertains to the final observation in the dataset. This is what we want so let's use actual data now in each of these cells.

```
actual <- matrix(data = rep(0, P * H), ncol = H)

for (i in 1:P) {
  first <- R + i
  last <- first + H - 1
  actual[i, 1:H] <- y$growth[first:last]
}
```

We are now going to create a similar matrix to store the forecasts from the ARMA(1,1) model. This matrix would have the following dimensions.

```
arma_fore <- matrix(data = rep(0, P * H), ncol = H)
```

Thereafter, we need to set the subsample for each ARMA(1,1) model. Estimate the parameters with the `arima` command and forecast eight steps ahead with the `predict` command. The results will then be stored in the object `ar_fore`.

```
for (i in 1:P) {
  arma_fore[i, 1:H] <- y %>%
    slice_head(n = (R + i - 1)) %>%
    pull(growth) %>%
    arima(., order = c(1, 0, 1)) %>%
    predict(., n.ahead = H) %>%
    pluck(1)
}
```

To compare these to the forecasts of an AR(1) model we create a similar matrix `ar_fore`.

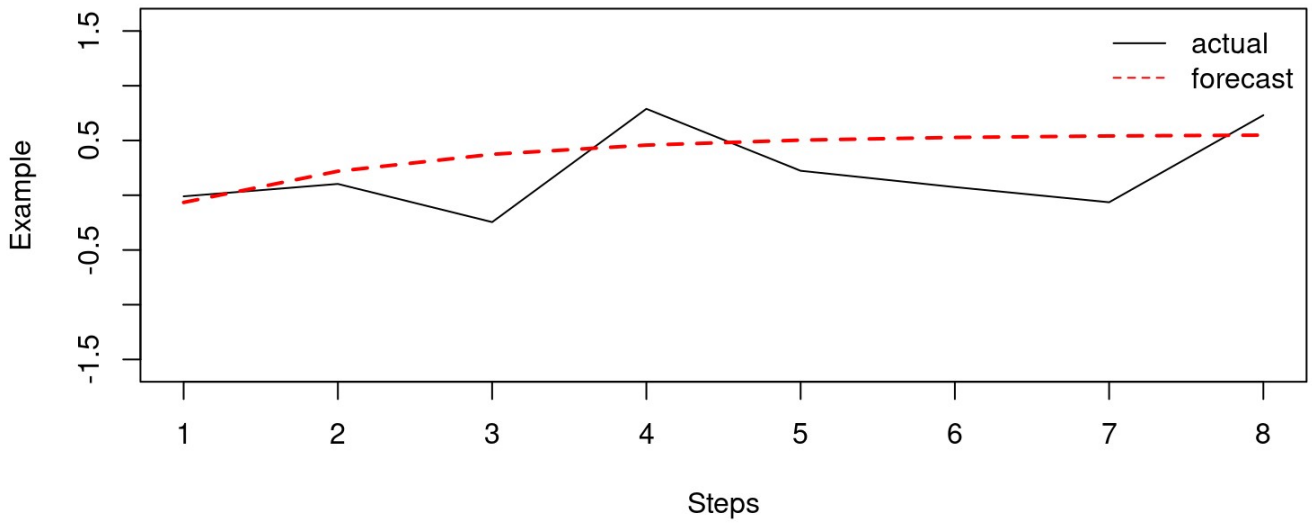
```
ar_fore <- matrix(data = rep(0, P * H), ncol = H)
```

Thereafter, we proceed as before:

```
for (i in 1:P) {
  ar_fore[i, 1:H] <- y %>%
    slice_head(n = (R + i - 1)) %>%
    pull(growth) %>%
    arima(., order = c(1, 0, 0)) %>%
    predict(., n.ahead = H) %>%
    pluck(1)
}
```

To take a look at one of the one- to eight-step ahead forecasts, we can look at what is going on at a particular row of each of these matrices. In this case we have selected row thirty.

```
rowf <- 30

par(mfrow = c(1, 1))
plot.ts(
  actual[rowf,],
  col = "black",
  ylab = "Example",
  xlab = "Steps",
  ylim = c(min(actual[rowf,] * (-2)), max(actual[rowf,] * 2))
)
lines(ar_fore[rowf,],
      col = "red",
      lty = 2,
      lwd = 2)
legend(
  "topright",
  legend = c("actual", "forecast"),
  lty = c(1, 2),
  col = c("black", "red"),
  bty = "n"
)
```



Note that when the horizon is relatively large, the model predictions are particularly poor. It is now relatively easy to create a matrix of similar dimensions that contains the forecast errors for each model.

```
error_arma <- actual - arma_fore
error_ar <- actual - ar_fore
```

To calculate the average bias at each step ahead, we simply have to find the mean values for each of

the columns of the `error_*` matrices.

```
bias_arma_step <-  colMeans(error_arma)
bias_ar_step <- colMeans(error_ar)
```

Note that one average, the model tends to provide a prediction that is too high over the short-run, but too low over the long-run. Similarly, to calculate the bias at each point in time for the average forecast horizon, we would simply need to find the mean values for each of the rows in the `error_*` matrix.

```
bias_arma_time <- rowMeans(error_arma)
bias_ar_time <- rowMeans(error_ar)
```

It is also just as easy to calculate the root-mean squared-error for each step ahead and over time with the aid of the following commands for the different forecast horizons.

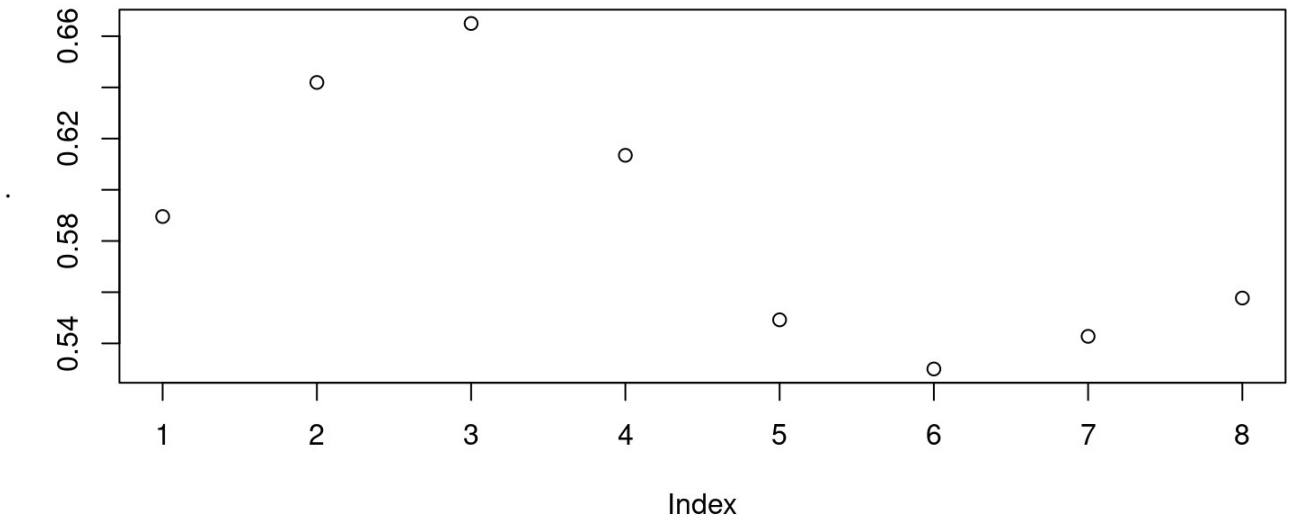
```
RMSE_arma_step <- sqrt(colMeans(error_arma ^ 2))
RMSE_ar_step <- sqrt(colMeans(error_ar ^ 2))
```

In addition, we could also consider how the RMSE for the average horizon may change over time.

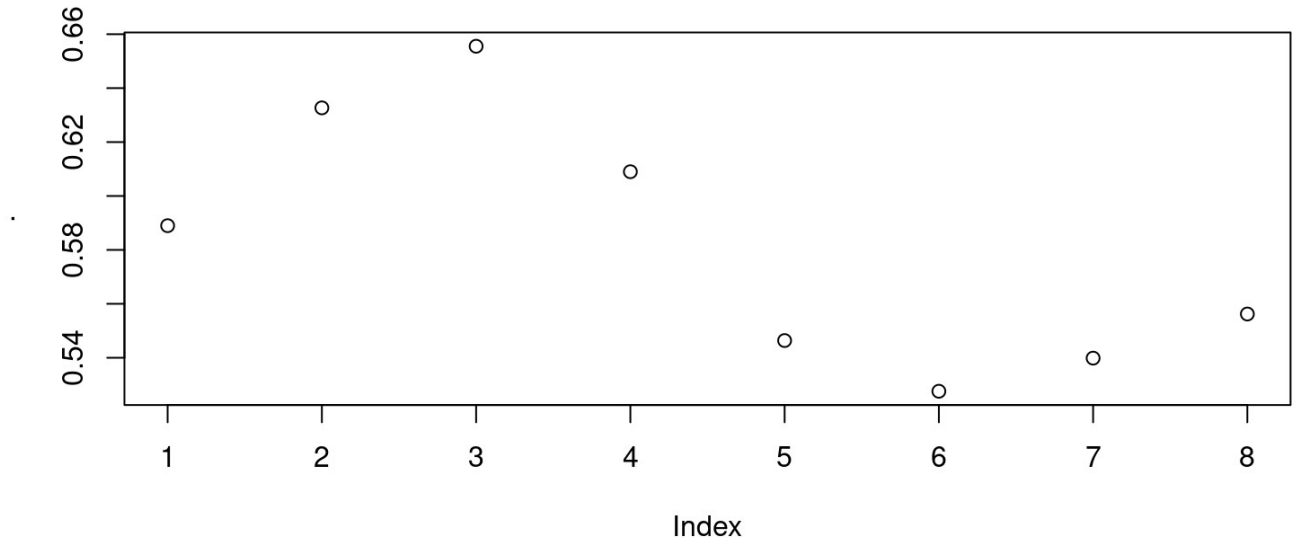
```
RMSE_arma_time <- sqrt(rowMeans(error_arma ^ 2))
RMSE_ar_time <- sqrt(rowMeans(error_ar ^ 2))
```

We could then display these results with the aid of a plot.

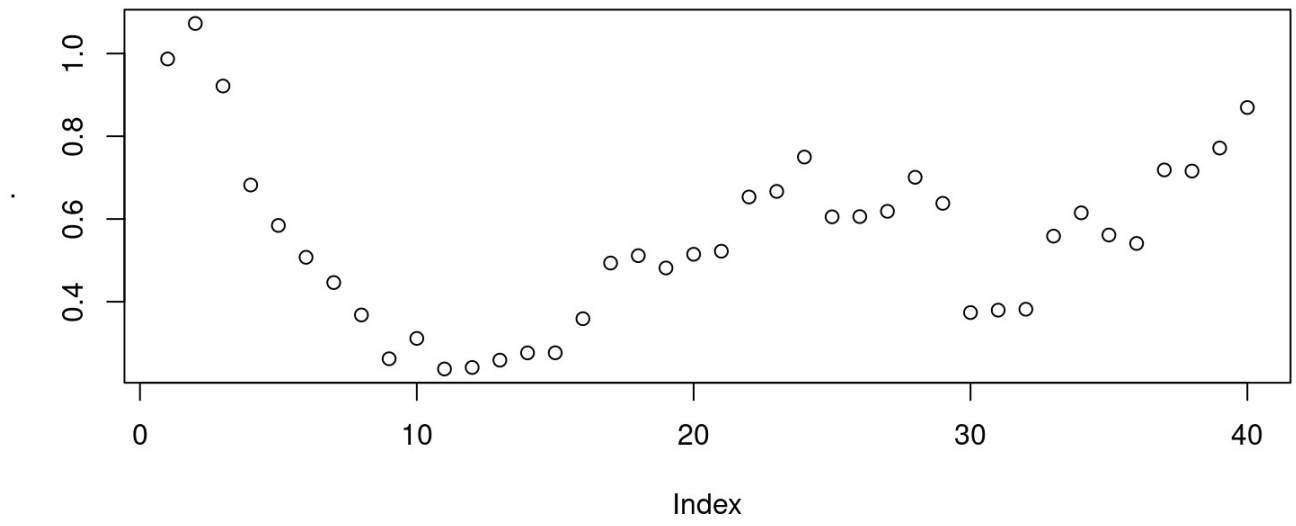
```
RMSE_arma_step %>%
  plot()
```



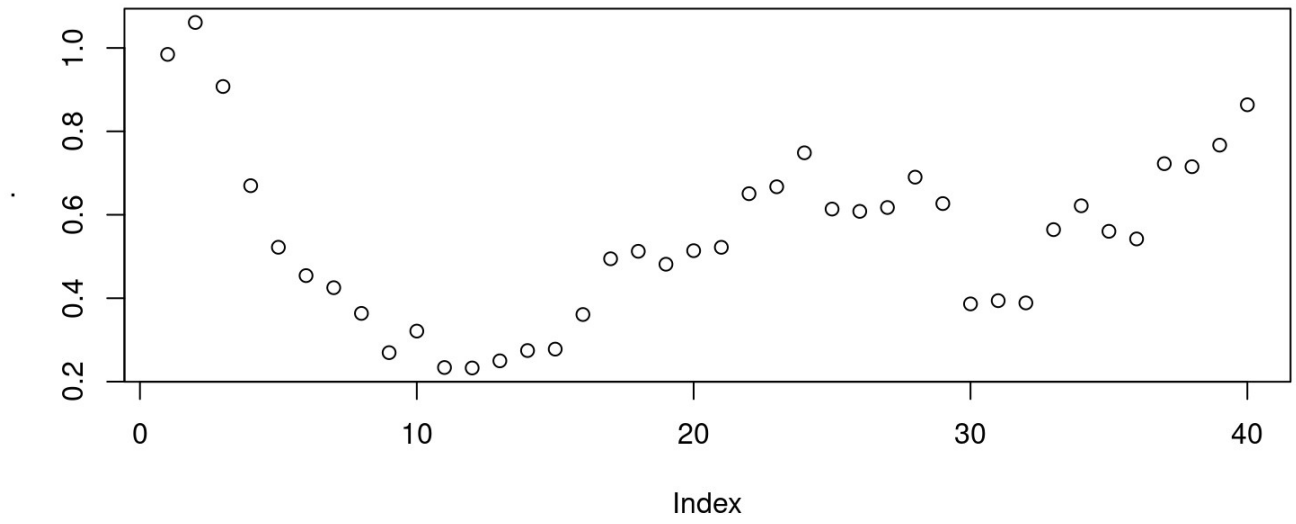
```
RMSE_ar_step %>%
  plot()
```




```
RMSE_arma_time %>%
  plot()
```




```
RMSE_ar_time %>%
  plot()
```



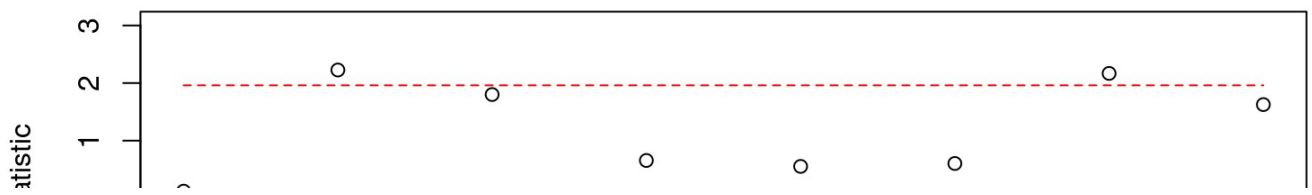
5 Model comparison

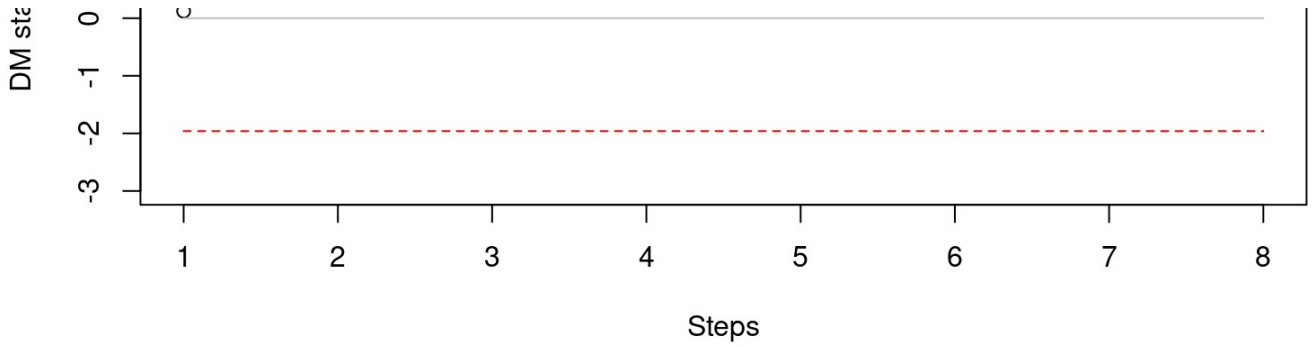
In what follows, we are going to compare the forecasts with the aid of the Diebold and Mariano (1995) statistic. To calculate this statistic for each step ahead, we can create a vector of eight observations and use the `dm.test` command that is contained in the `forecast` package.

```
dm_steps <- rep(0, H)

for (i in 1:H) {
  dm_steps[i] <-
    dm.test(error_arma[, i], error_ar[, i], h = i)$statistic
}
```

The easiest way to display these results is with the aid of a simple plot.

```
plot(dm_steps,
     ylim = c(-3, 3),
     xlab = "Steps",
     ylab = "DM statistic")
lines(rep(0, H), col = "grey", lty = 1)
lines(rep(1.96, H), col = "red", lty = 2)
lines(rep(-1.96, H), col = "red", lty = 2)
```





Note that most of the results are within the confidence bands, but the second and seventh observation clearly exceed the confidence interval, so in this case the forecasting results are not significantly different from one another. To determine which of these models provides the superior forecast, we could turn our attention to the RMSE values for these particular forecasts.

```
RMSE_arma_step[2]
```

```
## [1] 0.6419322
```

```
RMSE_ar_step[2]
```

```
## [1] 0.6326771
```

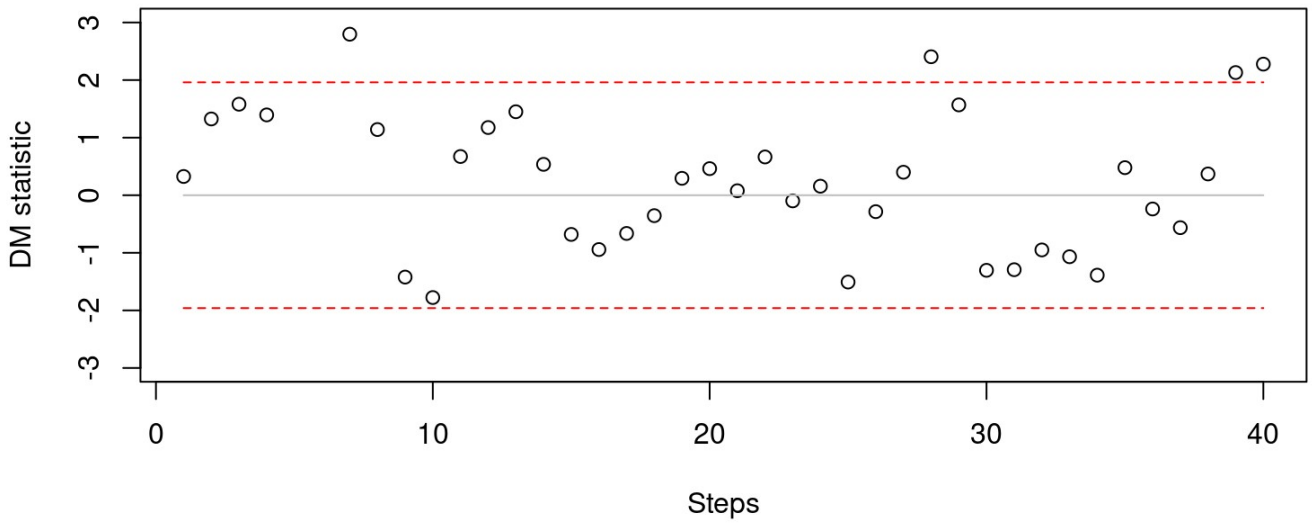
So the AR(1) provides the better forecast for $h = 2$ and $h = 7$, when compared to the ARMA(1,1). The difference in forecasting performance at this horizon is significantly different from zero.

We could also consider the Diebold and Mariano (1995) statistics for each period of time (although this is seldomly performed in most empirical studies). The commands that we use largely follow what has been described above.

```
dm_time <- rep(0, P)

for (i in 1:P) {
  dm_time[i] <-
    dm.test(error_arma[i,], error_ar[i,], h = 1)$statistic
}
```

To then plot the results:

```
plot(dm_time,
     ylim = c(-3, 3),
     xlab = "Steps",
     ylab = "DM statistic")
lines(rep(0, P), col = "grey", lty = 1)
lines(rep(1.96, P), col = "red", lty = 2)
lines(rep(-1.96, P), col = "red", lty = 2)
```



Almost all of the test statistics are within the critical values. To find out, which model provides the superior forecast when the values exceed the upper bound, we can look at the RMSE for observation number 5.

```
RMSE_arma_time[5]
```

```
## [1] 0.5844212
```

```
RMSE_ar_time[5]
```

```
## [1] 0.5221418
```

Again the AR(1) is better and note that at no time was it significantly worse, since none of the Diebold and Mariano (1995) statistics exceed the lower bound.

This would suggest that the ARMA(1,2) model is significantly better when the DM statistic is above +1.96 and the AR(1) model is significantly better when the DM statistic is below -1.96.

5.1 Comparing nested models

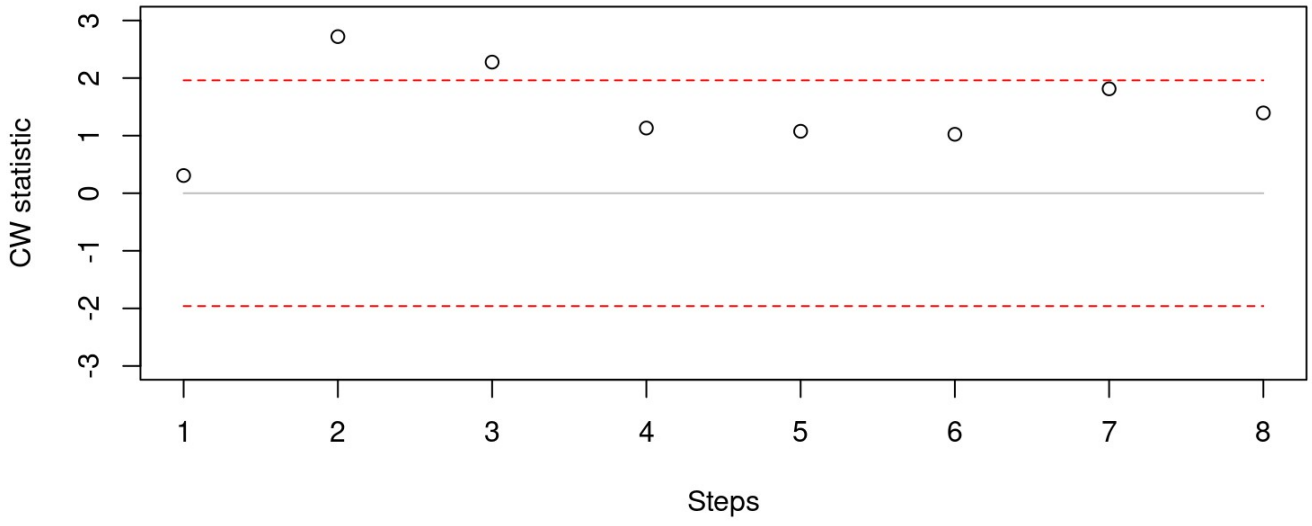
Since these models are nested within one another the Diebold and Mariano (1995) one would usually make use of the statistic of Clark and West (2007). However, the encompassing test is one-sided and is used to test if the bigger model provides superior forecasts. Therefore, the above results from the Diebold and Mariano (1995) test would be appropriate, since the encompassing test should not be used to test if the performance of the smaller model is significantly better than the larger model.

However, if we nevertheless want to use the Clarke-West statistic we could use the routines within the `tsm` package. The results for the statistic and corresponding *p*-value for each step ahead can be stored in vectors, as followings

```
cwstat_steps <- rep(0, H)
cwpval_steps <- rep(0, H)

for (i in 1:H) {
  cwstat_steps[i] <-
    cw(error_arma[, i], error_ar[, i], arma_fore[, i], ar_fore[, i])$test
  cwpval_steps[i] <-
    cw(error_arma[, i], error_ar[, i], arma_fore[, i], ar_fore[, i])$pvalue
}
```

The results of this test can be plotted as before, where:

```
plot(cwstat_steps,
     ylim = c(-3, 3),
     xlab = "Steps",
     ylab = "CW statistic")
lines(rep(0, H), col = "grey", lty = 1)
lines(rep(1.96, H), col = "red", lty = 2)
lines(rep(-1.96, H), col = "red", lty = 2)
```

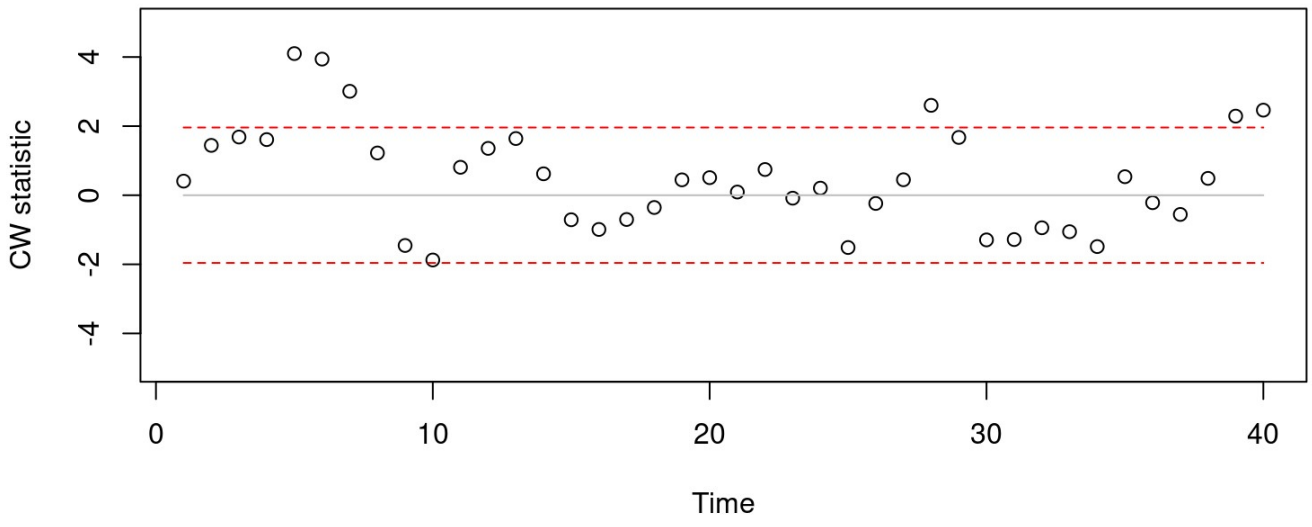


Note that since we have not applied this test in an appropriate setting we should not interpret the results. If we wanted to perform these tests over time, we could proceed as follows:

```
cwstat_time <- rep(0, P)
cwpval_time <- rep(0, P)

for (i in 1:P) {
  cwstat_time[i] <-
    cw(error_arma[i,], error_ar[i,], arma_fore[i,], ar_fore[i,])$test
  cwpval_time[i] <-
    cw(error_arma[i,], error_ar[i,], arma_fore[i,], ar_fore[i,])$test
}
```

And these results maybe displayed as follows:

```
plot(cwstat_time,
     ylim = c(-5, 5),
     xlab = "Time",
     ylab = "CW statistic")
lines(rep(0, P), col = "grey", lty = 1)
lines(rep(1.96, P), col = "red", lty = 2)
lines(rep(-1.96, P), col = "red", lty = 2)
```



which we would be unable to interpret once again.

5.2 Other forecast plots

To plot a hairy plot one could make use of the following set of commands.

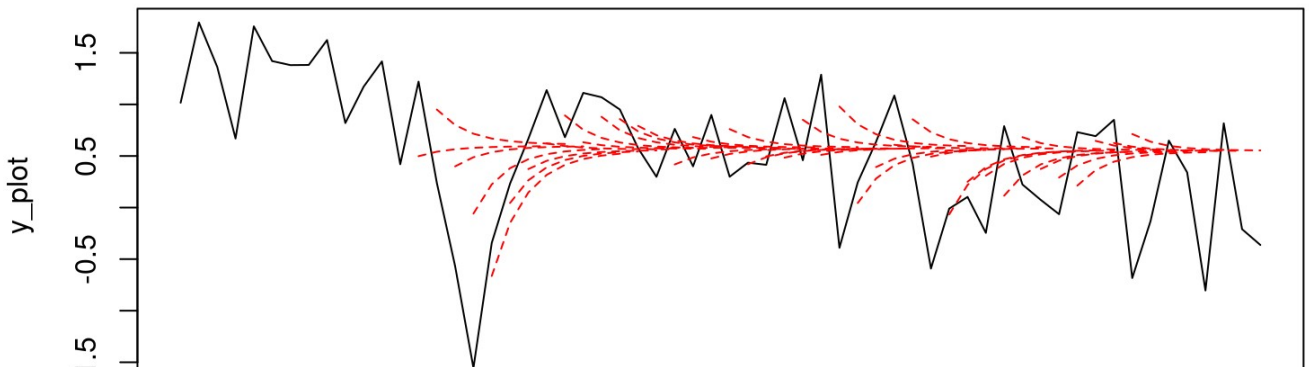
```
h_start <- R - 12

y_plot <-
  ts(y$growth[h_start:length(y$growth)], start = c(2005, 1), frequency =
       4)

n_obs <- length(y_plot)
no_fore <- n_obs - P - H
nons <- n_obs - H

fore_plot <- matrix(data = rep(0, n_obs * P), nrow = P)

for (i in 1:P) {
  nan_beg <- rep(NaN, no_fore + i)
  nan_end <- rep(NaN, P - i)
  fore_plot[i, 1:n_obs] <- c(nan_beg, ar_fore[i,], nan_end)
}

length(fore_plot[i, 1:n_obs])
```

```
## [1] 60
```

```
length(c(nan_beg, ar_fore[i,], nan_end))
```

```
## [1] 60
```

```
plot.ts(y_plot)

for (i in 1:P) {
  lines(ts(fore_plot[i,], start = c(2005, 1), frequency = 4),
        col = "red",
        lty = 2)
}
```



6 Constructing a forecast density

The first thing that we would need to do when constructing a forecast density is to set the horizon and intervals. In this case we are going to make use of a sixteen-step ahead forecast and percentiles that take on specific values.

```
H <- 16
percentiles <- c(0.10, 0.15, 0.25, 0.35)
num <- dim(y)[1]
inter <- length(percentiles) * 2
```

We then need to estimate the model parameters and generate a forecast.

```
ar_mod <- tibble(forecast = y %>%
                   pull(growth) %>%
                   arima(., order = c(1, 0, 0)) %>%
                   predict(., n.ahead = H) %>%
                   pluck(1))
```

To calculate the density, we make use of information from the forecast mean squared error. These values are all going to be contained in the column vector `mse` .

```
ar_mod <- ar_mod %>%
  mutate(mse = rep(0, H))

for (i in 1:H) {
  ar_mod$mse[i] <- var(c(y$growth, ar_mod$forecast[i]))
}
```

Now we need to prepare the fan chart.

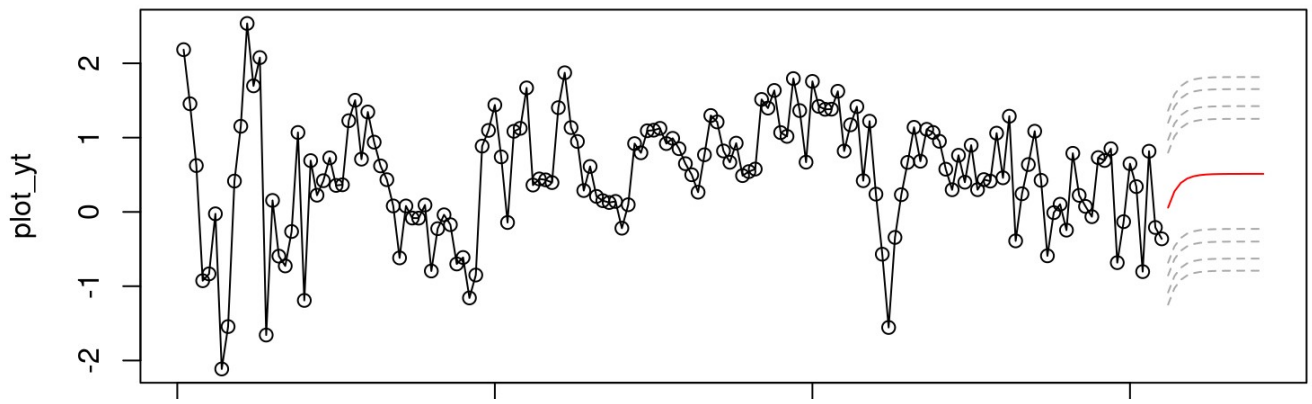
```
fanChart <- matrix(rep(0, H * inter), nrow = H)

for (i in 1:length(percentiles)) {
  z2 <- abs(qnorm(percentiles[i] / 2, 0, 1))
  forcInt <- matrix(rep(0, H * 2), ncol = 2)

  for (h in 1:H) {
    forcInt[h, 1] <- ar_mod$forecast[h] - z2 * sqrt(ar_mod$mse[h])
    forcInt[h, 2] <- ar_mod$forecast[h] + z2 * sqrt(ar_mod$mse[h])

    fanChart[h, i] <- forcInt[h, 1]
    fanChart[h, inter - i + 1] <- forcInt[h, 2]
  }
}
```

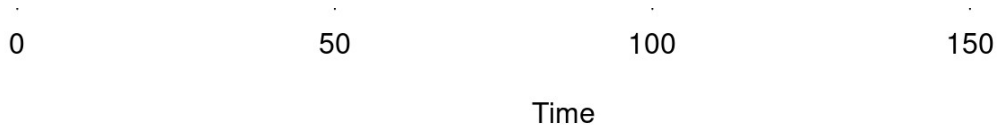
We can then create objects for the respective elements that will find their way onto the plot.

```
plot_len <- num + H

plot_yt <- c(y$growth, rep(NaN, H))
plot_ytF <- c(rep(NaN, num), ar_mod$forecast)
```

The point forecast estimate can then be plotted, with the corresponding predictive density.

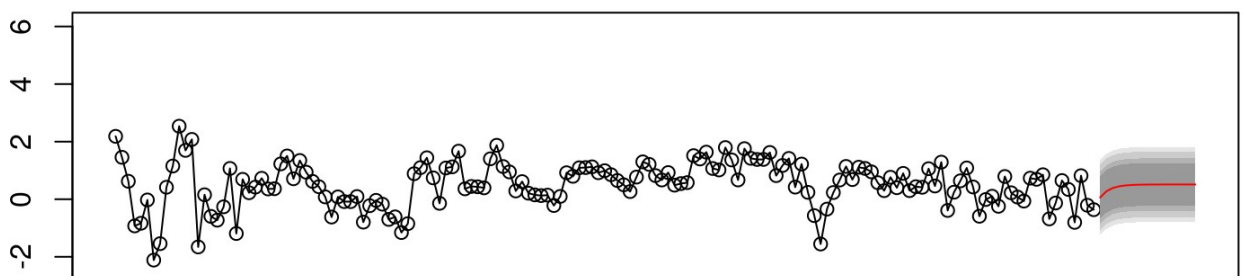
```
plot.ts(plot_yt, type = "o")
lines(plot_ytF, col = "red", lty = 1)

plot_ytD <- rbind(matrix(rep(NaN, num * inter), ncol = inter), fanChart)

no_int <- dim(plot_ytD)

for (i in 1:no_int[2]) {
  lines(plot_ytD[, i], col = "darkgrey", lty = 2)
}
```





As this does not look very professional, we can make use of the shading with the `polygon` command in what follows.

```
start_obs <- num + 1

plot(
  plot_yt,
  type = "o",
  ylim = c(-6, 6),
  ylab = "",
  xlab = ""
)

polygon(c(c(start_obs:plot_len), rev(c(start_obs:plot_len))),
        c(plot_ytD[start_obs:plot_len, 1], rev(plot_ytD[start_obs:plot_len, 8])),
        col = "grey90",
        border = NA)

polygon(c(c(start_obs:plot_len), rev(c(start_obs:plot_len))),
        c(plot_ytD[start_obs:plot_len, 2], rev(plot_ytD[start_obs:plot_len, 7])),
        col = "grey80",
        border = NA)

polygon(c(c(start_obs:plot_len), rev(c(start_obs:plot_len))),
        c(plot_ytD[start_obs:plot_len, 3], rev(plot_ytD[start_obs:plot_len, 6])),
        col = "grey70",
        border = NA)

polygon(c(c(start_obs:plot_len), rev(c(start_obs:plot_len))),
        c(plot_ytD[start_obs:plot_len, 4], rev(plot_ytD[start_obs:plot_len, 5])),
        col = "grey60",
        border = NA)

lines(plot_ytF, col = "red", lty = 1)
```



7 References

Bai, Jushan, and Pierre Perron. 2003. “Computation and Analysis of Multiple Structural Change Models.” *Journal of Applied Econometrics* 18 (1): 1–22.

Clark, Todd E., and Kenneth D. West. 2007. “Approximately Normal Tests for Equal Predictive Accuracy in Nested Models.” *Journal of Econometrics* 138: 291–311.

Diebold, Francis X., and Roberto S. Mariano. 1995. “Predictive Accuracy.” *Journal of Business and Economic Statistics* 13 (3): 253–63.