Non-seasonal Data - High Tech Patent Applications in the EU

1. Exploratory Data Analysis

1.1. Data Source, Headers and Labels

```r
#set working directory
setwd("~/DIT/Time Series 2 Forecasting/Project/1-Trend only")

#Graphical Parameters: For colours, color specifications, check colors() or, even better, demo(colors)

library(readr)
library(forecast)
library(tseries)

patentEu28Data <- read_csv("pat_ep_ntec/pat_ep_ntec_1_Data.csv",
                          col_types = cols(Value = col_number()))

names(patentEu28Data) #check column names
```

```
## [1] "GEO"                 "TIME"                "IPC"
## [4] "UNIT"                "Value"               "Flag and Footnotes"
```

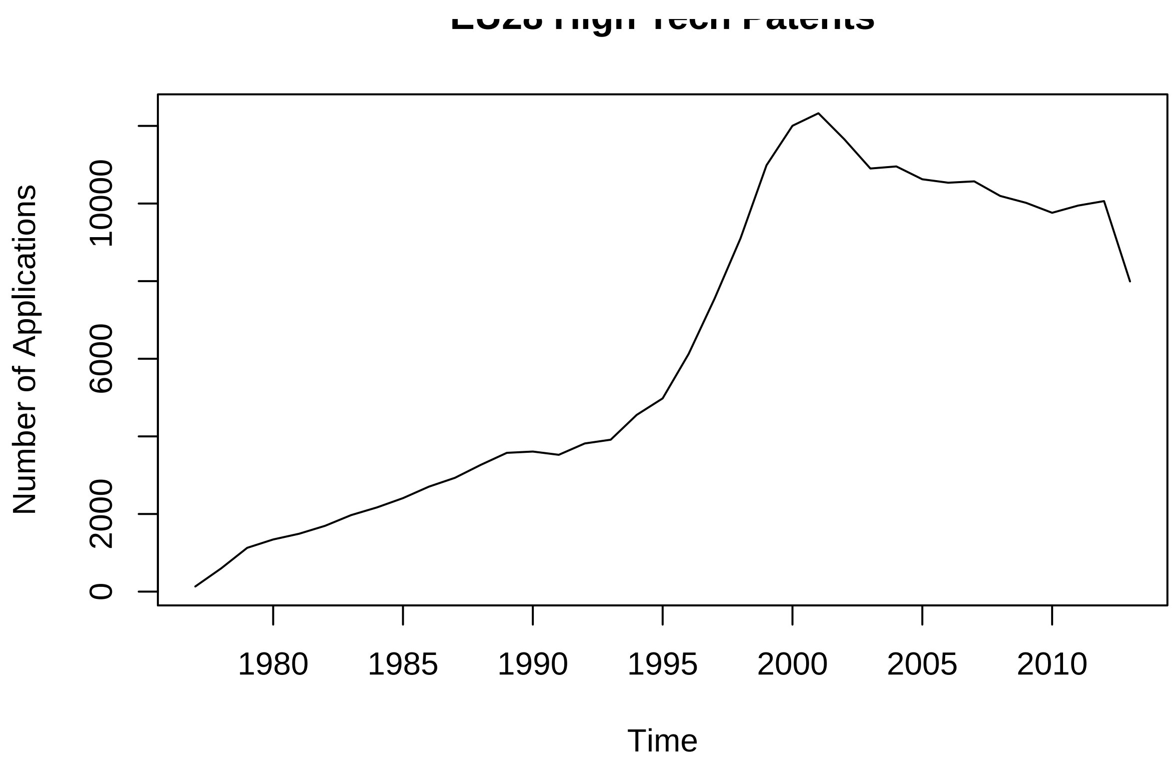
Plotting the Time Series

```r
head(patentEu28Data[2], 1) #check starting time
```

```r
values = patentEu28Data[5]
```

The starting date for this time series is 1977.

```r
values = ts(values, start=1977, frequency=1)
ts.plot(values, main="EU28 High Tech Patents", ylab="Number of Applications", type="l")
```




```r
values
```

```
## Time Series:
## Start = 1977
## End = 2013
## Frequency = 1
##          Value
##  [1,]   132.00
##  [2,]   599.78
##  [3,]  1128.32
##  [4,]  1343.04
##  [5,]  1491.16
##  [6,]  1695.70
##  [7,]  1970.68
##  [8,]  2169.94
##  [9,]  2408.72
## [10,]  2704.74
## [11,]  2931.51
## [12,]  3267.64
## [13,]  3575.93
## [14,]  3609.14
## [15,]  3525.19
## [16,]  3817.37
## [17,]  3914.17
## [18,]  4553.05
## [19,]  4979.21
## [20,]  6123.73
## [21,]  7550.26
## [22,]  9104.85
## [23,] 10985.20
```

```
## [24,] 12003.43
## [25,] 12325.98
## [26,] 11653.36
## [27,] 10901.80
## [28,] 10955.94
## [29,] 10624.97
## [30,] 10536.50
## [31,] 10571.42
## [32,] 10195.10
## [33,] 10017.04
## [34,]  9761.64
## [35,]  9946.93
## [36,] 10062.58
## [37,]  7991.98
```

Separating the data into a "training set" with the values up until 2010, and holding out the last three observed values to run diagnostics on the accuracy of a chosen model:

```r
start(values)
```

```
## [1] 1977    1
```

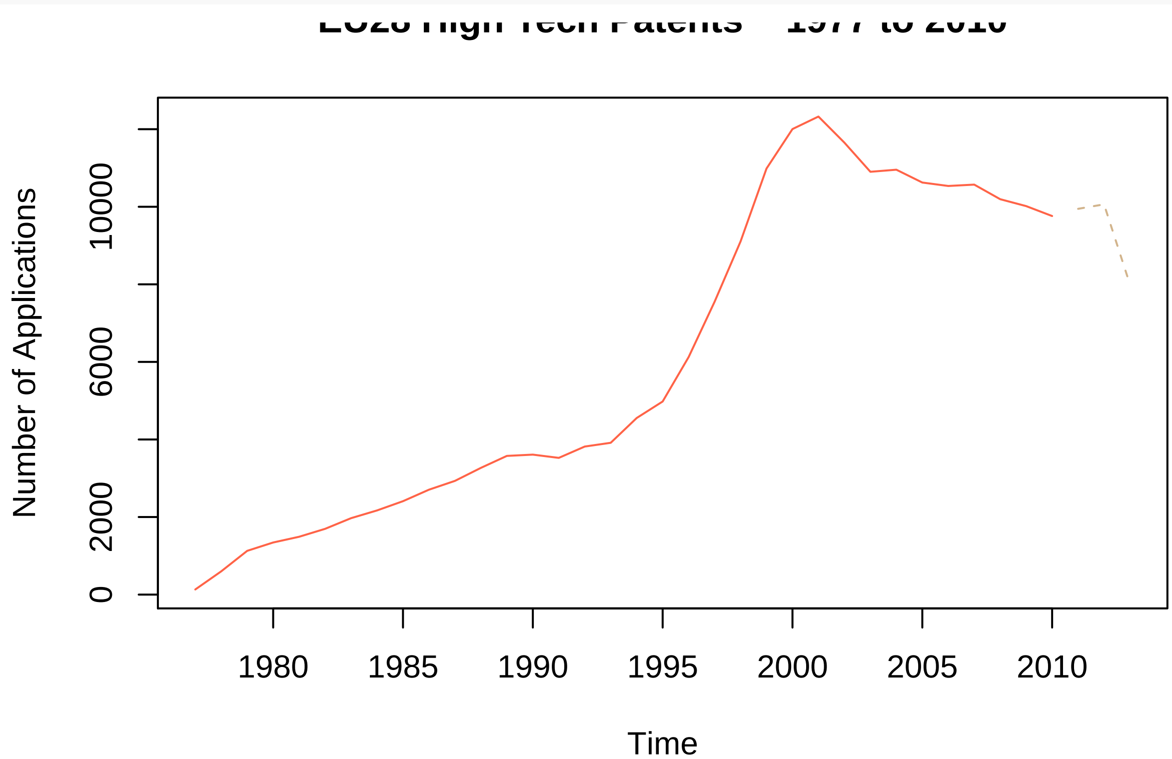
```r
end(values)
```

```
## [1] 2013    1
```

```r
values.holdout <- window(values, start=2011, end=2013)
values <- window(values, end=2010)
```

Visually we now have:

```r
ts.plot(cbind(values, values.holdout), main="EU28 High Tech Patents - 1977 to 2010",
        ylab="Number of Applications", type="l", col=c("tomato", "tan"), lty=c(1, 2))
```



Checking for missing values:

```
#check if there are missing values
complete <- TRUE
for(c in complete.cases(values)) {
  if(!c){
    complete == FALSE
  }
}
if(complete){
  print("No missing values")
} else {
  print ("There are missing values, use omit NA")
}
```

```
## [1] "No missing values"
```

2. Is this data stationary?

The stationarity property of the data (before or after transforms) will determine how we can model it. For models like ARIMA it's required that the data can be made stationary.

The data doesn´t appear to be stationary - just by observing the plot, it is apparent that average and variance change over time. So we need to transform our series. Some of the possible mathematical transforms include: differencing, log (and Box-Cox), moving average, percent change, lag, or cumulative sum.

Checking stationarity using ADF and KPSS tests

We can more accurately check if the time series is stationary by using the Augmented Dickey-Fuller Test (adf test). A p-Value of less than 0.05 in adf.test() indicates that it is stationary. KPSS test is used in complement to ADF. If the result from both tests suggests that the time series in stationary, then it probably is.

> "KPSS-type tests are intended to complement unit root tests, such as the Dickey–Fuller tests. By testing both the unit root hypothesis and the stationarity hypothesis, one can distinguish series that appear to be stationary, series that appear to have a unit root, and series for which the data (or the tests) are not sufficiently informative to be sure whether they are stationary or integrated."

KPSS reference: D. Kwiatkowski et al., Testing the null hypothesis of trend stationarity (1992)

PP tests

Alternatively or additionally, test for the null hypothesis that the series has a unit root (alternative hypothesis being that it is stationary). Integrates DF test.

```
#library(tseries)
adf.test(values) # p-value < 0.05 indicates the TS is stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values
## Dickey-Fuller = -1.8489, Lag order = 3, p-value = 0.6315
## alternative hypothesis: stationary
```

```
kpss.test(values, null="Trend") # trend/level stationarity test
```

```
## Warning in kpss.test(values, null = "Trend"): p-value greater than printed
## p-value
```

```
##
##  KPSS Test for Trend Stationarity
##
## data:  values
## KPSS Trend = 0.096859, Truncation lag parameter = 3, p-value = 0.1
```

```r
kpss.test(values, null="Level")
```

```
## Warning in kpss.test(values, null = "Level"): p-value smaller than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  values
## KPSS Level = 0.85858, Truncation lag parameter = 3, p-value = 0.01
```

```r
pp.test(values, lshort = FALSE)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  values
## Dickey-Fuller Z(alpha) = -5.1384, Truncation lag parameter = 9,
## p-value = 0.8049
## alternative hypothesis: stationary
```

ADF Null hypothesis: Time series is not stationary. ADF test shows we can't reject the null, this time-series is not stationary.

KPSS Null="Trend": The time series is trend-stationary. KPSS, with null hypothesis that trend is stationary, returns a p-value of 0.08, so we don't reject the null. It tells us that this series is trend-stationary - the data is stationary around the trend, it follows a straight line time trend with stationary errors. If the series is level stationary, it is akin to a random walk.

KPSS Null="Level": The time series stationary. The p-value is below 0.01, so we reject that this series is stationary.

So far, this is on par with our first intuition looking at the plot and in-line with dealing with real world untransformed series data.

2.1. Transforms

Difference

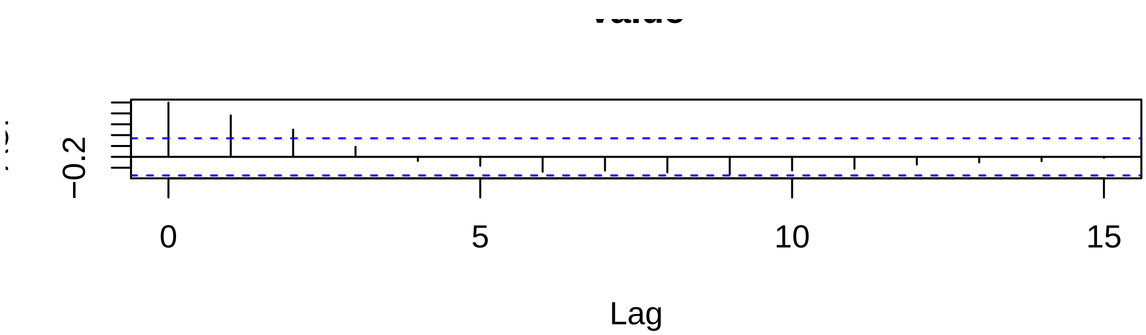
```r
values_diff1 = diff(values, lag = 1)
values_diff2 = diff(values_diff1)
tdiff <- cbind(values, values_diff1, values_diff2)
plot(tdiff, main="Differencing")
```

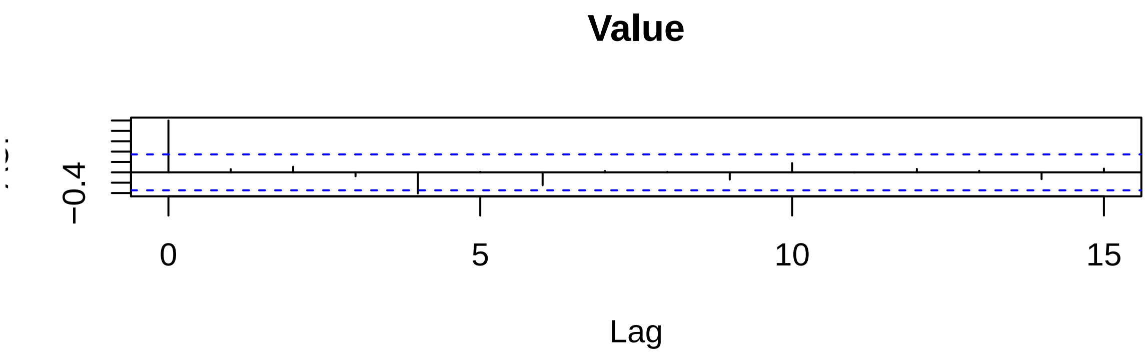
Maybe a second difference, looking at adf and acf we can confirm this:

```
par(mfrow=c(2,1))
acf(values_diff1)
adf.test(values_diff1)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values_diff1
## Dickey-Fuller = -2.5951, Lag order = 3, p-value = 0.3429
## alternative hypothesis: stationary
```

```
acf(values_diff2)
```

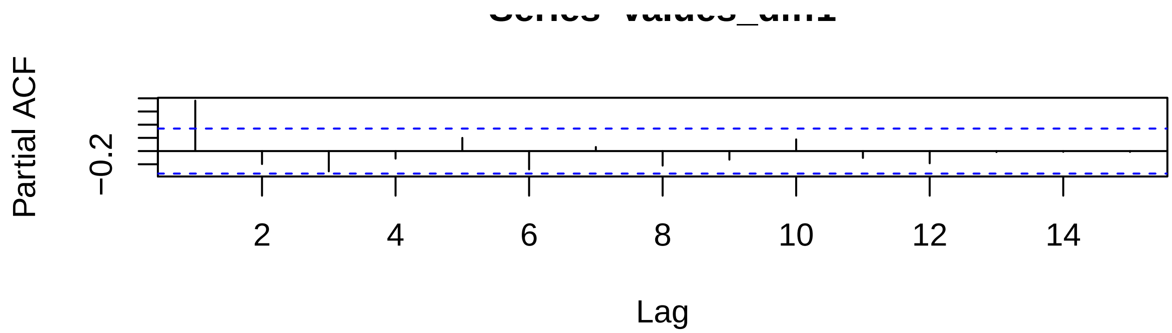


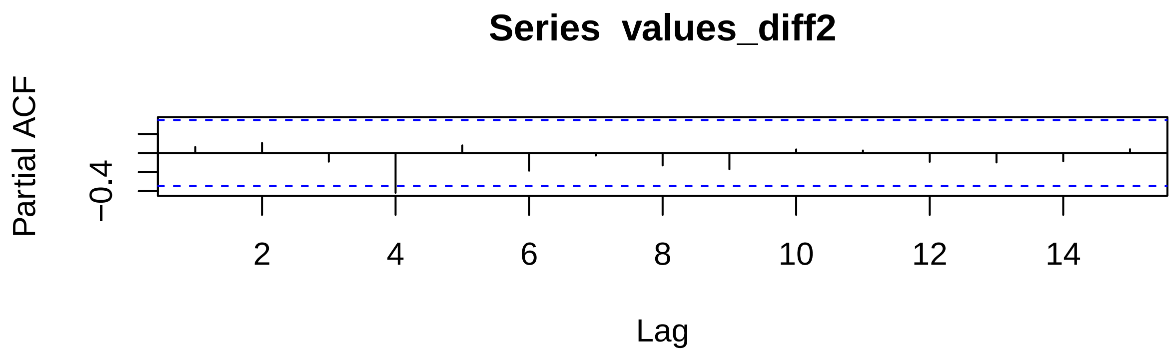



```
adf.test(values_diff2)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values_diff2
## Dickey-Fuller = -3.7642, Lag order = 3, p-value = 0.03608
## alternative hypothesis: stationary
```

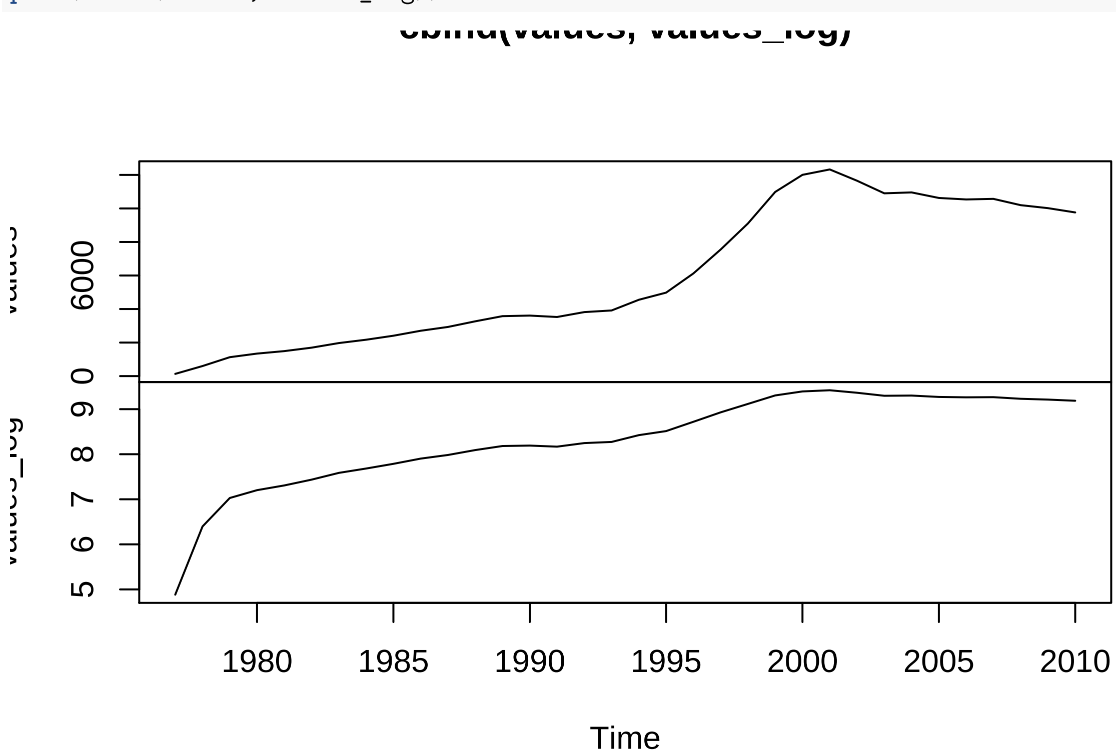
```
par(mfrow=c(2,1))
pacf(values_diff1)
pacf(values_diff2)
```





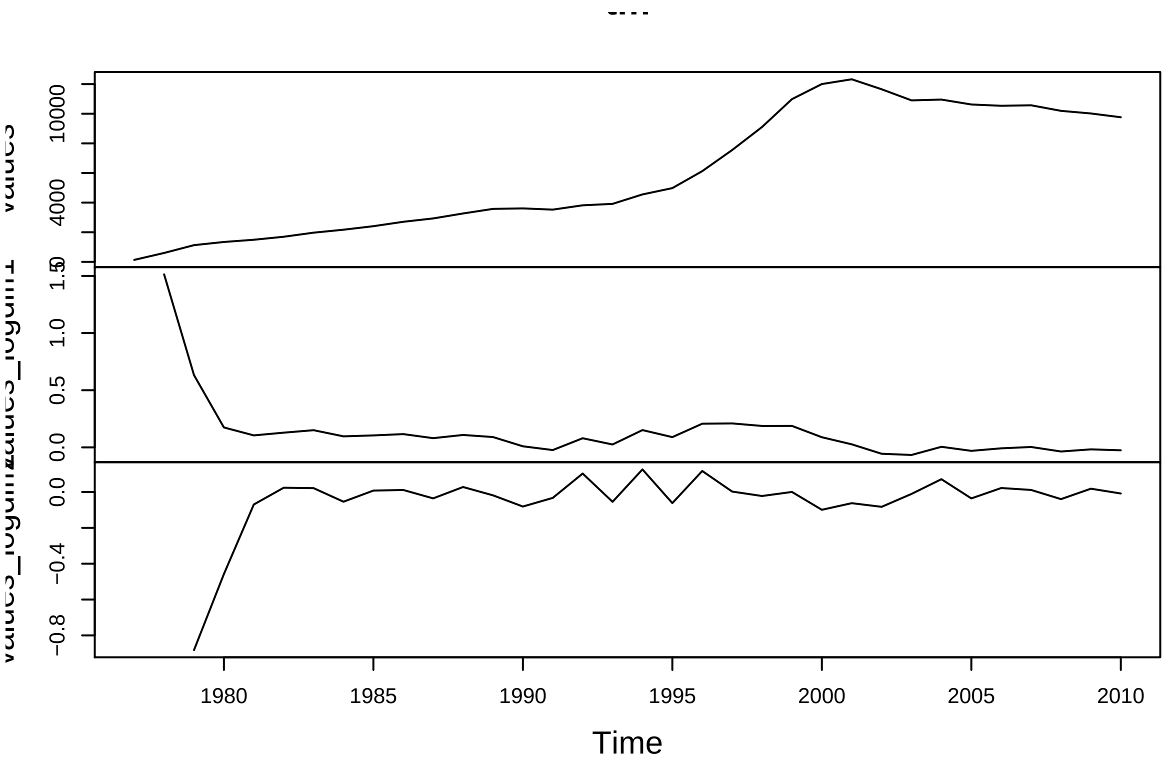
Log

```
values_log = log(values)
plot(cbind(values, values_log))
```



Log Difference

```
values_logdiff1 = diff(log(values), lag = 1)
values_logdiff2 = diff(values_logdiff1, lag = 1)
tm <- cbind(values, values_logdiff1, values_logdiff2)
plot(tm)
```

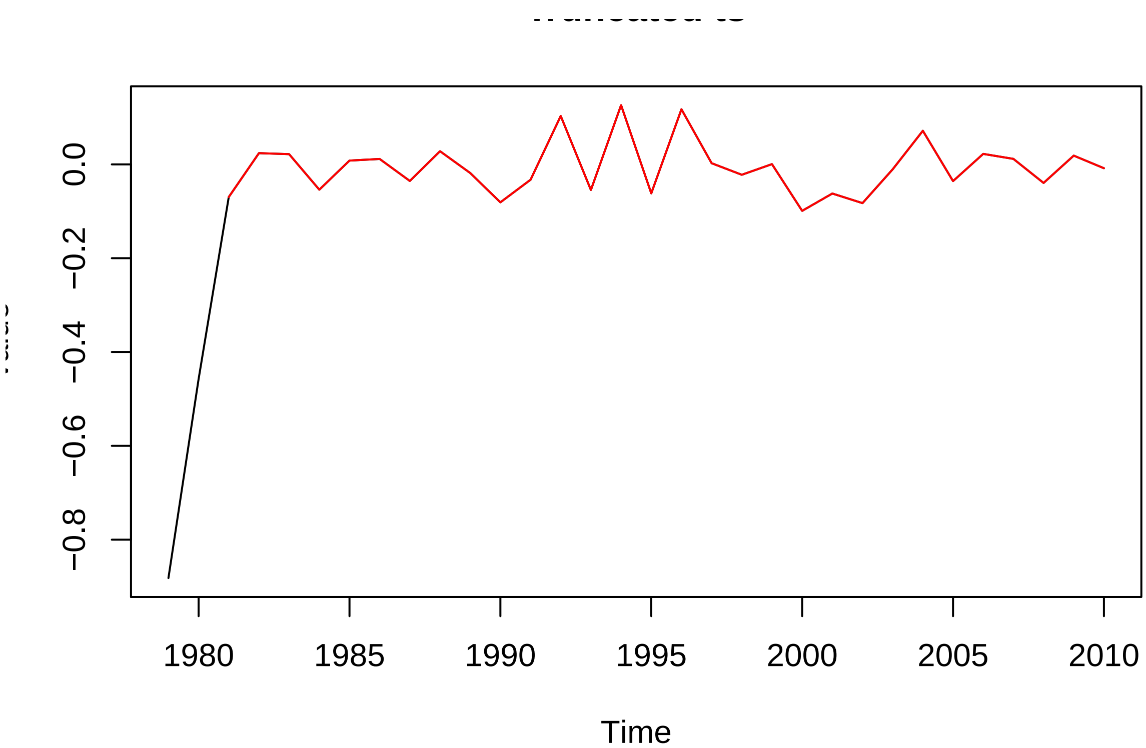


Looking at the above, ***we can consider that taking the second difference of the values might be enough.***. In this case, having more values would actually be detrimental to our analysis and forecast.

```
plot(values_logdiff2, main="Truncated ts")
values.logdiff.trunc <- window(values_logdiff2, start=1981, end=2012)
```

```
## Warning in window.default(x, ...): 'end' value not changed
```

```
lines(values.logdiff.trunc, col="red")
```



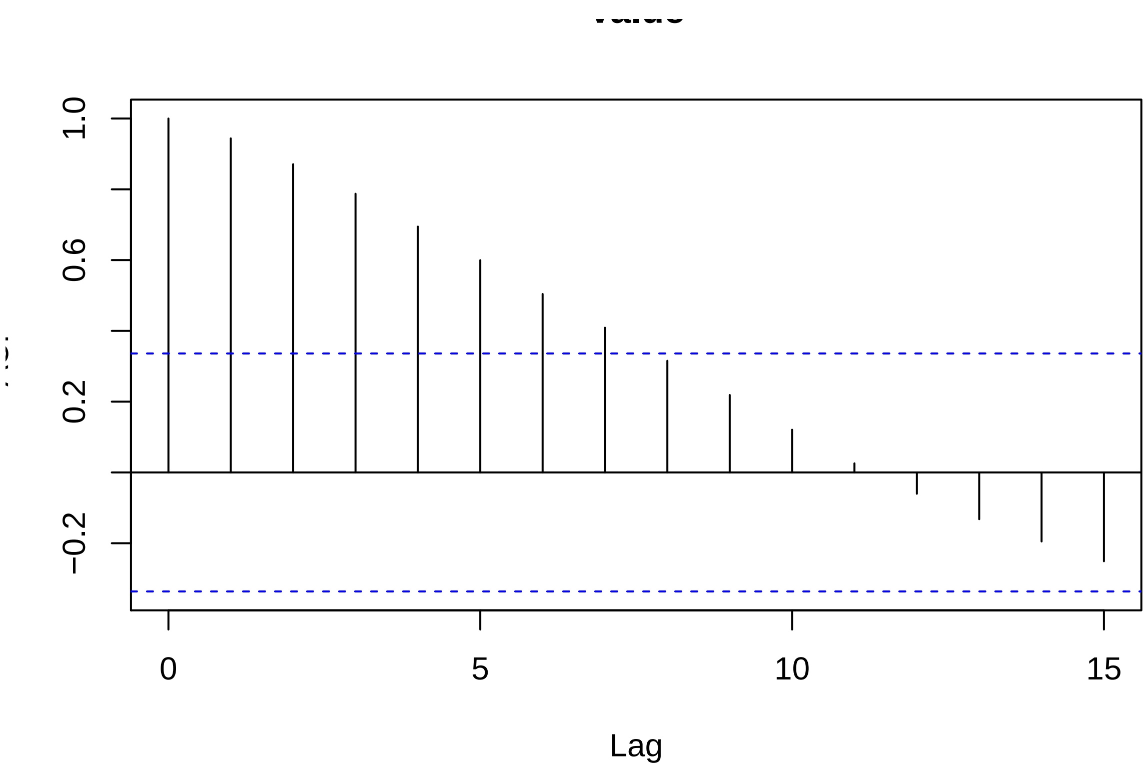
2.2 Check stationarity of different transforms

We can check if these transformations rendered the original time series values stationary, by checking the ACF and/or using ADF and KPSS again.

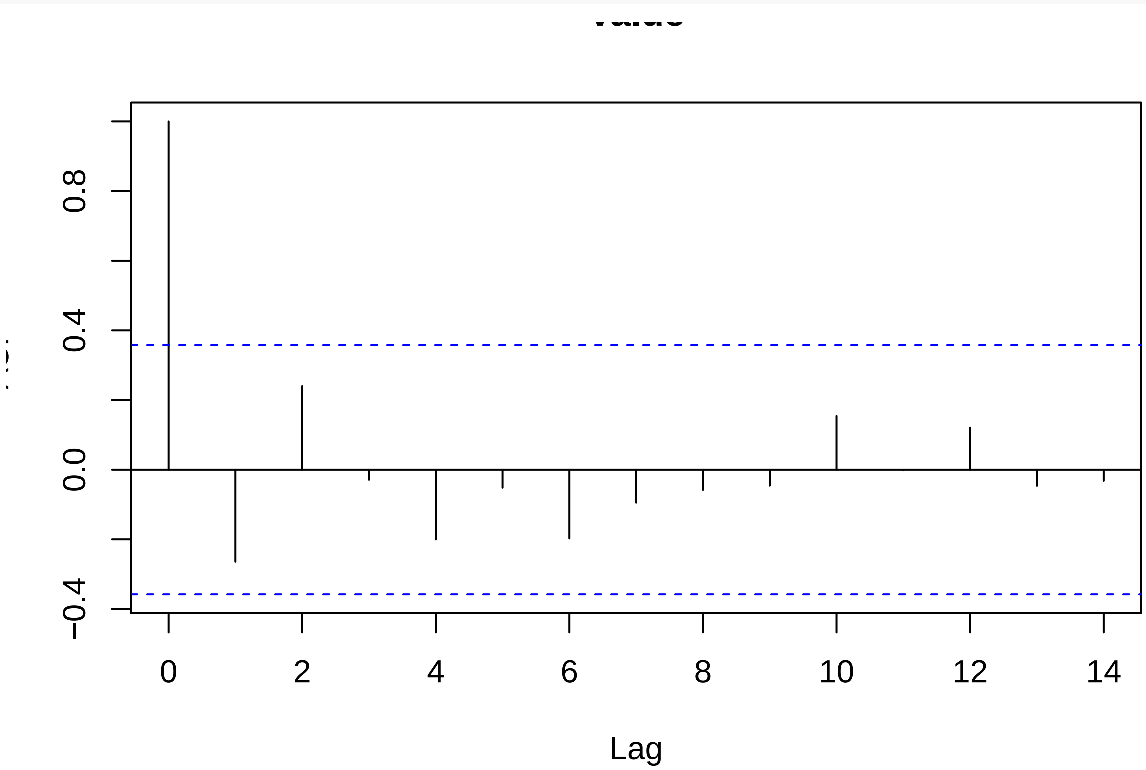
Using ACF

A stationary time series will have the autocorrelation fall to zero fairly quickly but for a non-stationary series it drops gradually.

```
acf(values)
```




```
acf(values.logdiff.trunc)
```



Using ACF, values.logdiff.trunc seems to be stationary

Using KPSS, ADF and unit root tests

Let's look at KPSS and ADF

```r
adf.test(values.logdiff.trunc, k=1)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values.logdiff.trunc
## Dickey-Fuller = -3.2245, Lag order = 1, p-value = 0.1023
## alternative hypothesis: stationary
```

```r
kpss.test(values.logdiff.trunc)
```

```
## Warning in kpss.test(values.logdiff.trunc): p-value greater than printed p-
## value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  values.logdiff.trunc
## KPSS Level = 0.06296, Truncation lag parameter = 2, p-value = 0.1
```

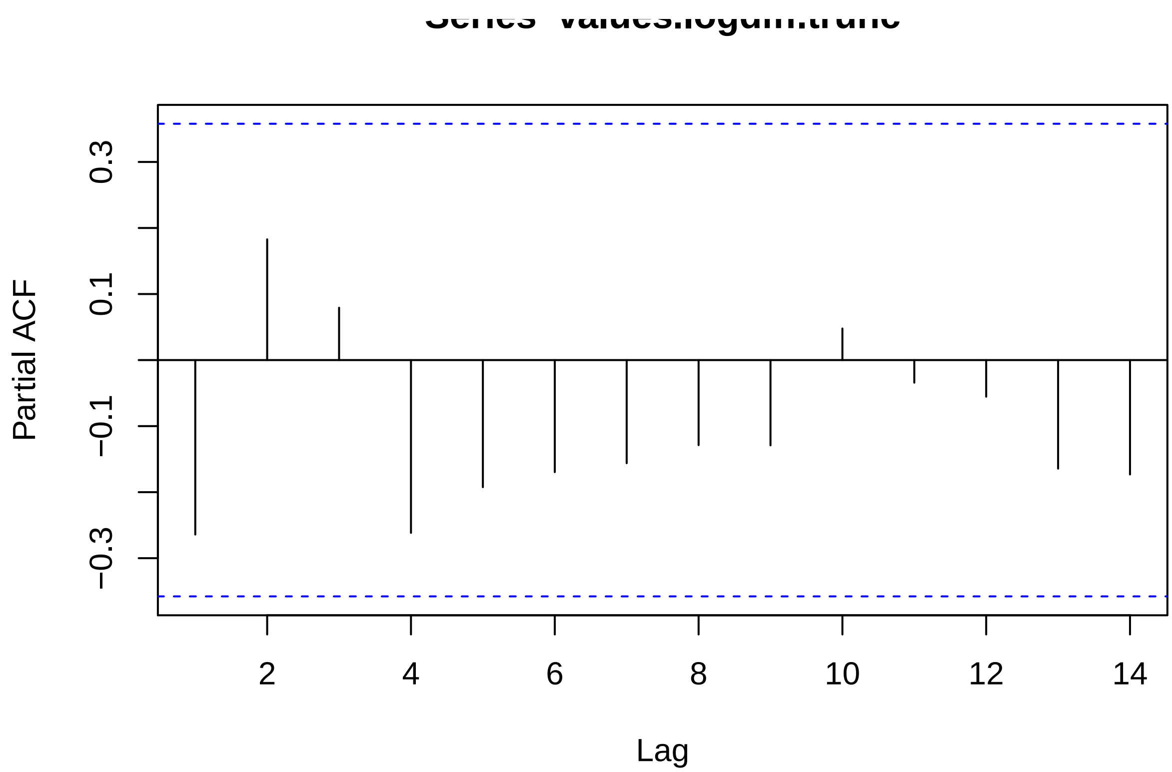
```r
pp.test(values.logdiff.trunc, alternative="stationary")
```

```
## Warning in pp.test(values.logdiff.trunc, alternative = "stationary"): p-
## value smaller than printed p-value
```

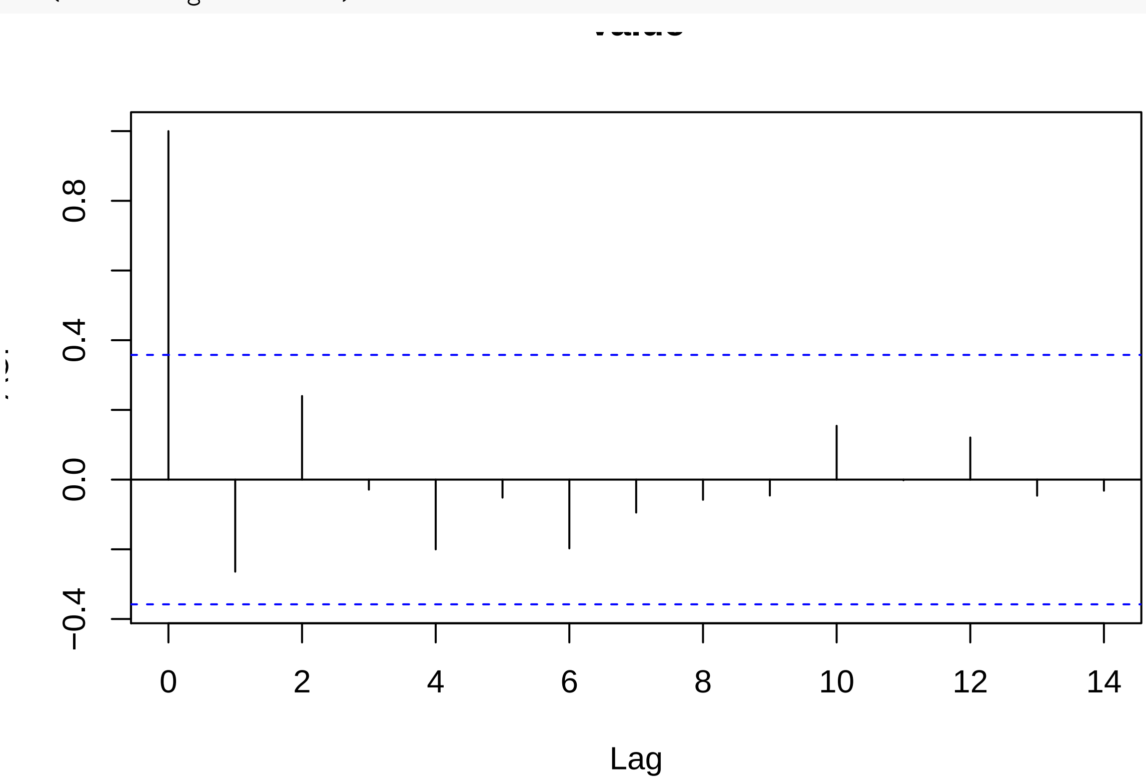
```
##
##  Phillips-Perron Unit Root Test
##
## data:  values.logdiff.trunc
## Dickey-Fuller Z(alpha) = -39.562, Truncation lag parameter = 2,
## p-value = 0.01
## alternative hypothesis: stationary
```

Although ADF is not returning a significant value for the stationarity hypothesis, Philip-Perron and KPSS tests indicate this time series to be stationary.

```r
pacf(values.logdiff.trunc)
```




```
acf(values.logdiff.trunc)
```



TODO: Test KPSS over a range of lags Examples

```
x <- rnorm(1000)  # is level stationary
kpss.test(x)
```

```
## Warning in kpss.test(x): p-value greater than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  x
## KPSS Level = 0.34527, Truncation lag parameter = 7, p-value = 0.1
```

```r
y <- cumsum(x)  # has unit root
kpss.test(y)
```

```
## Warning in kpss.test(y): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  y
## KPSS Level = 8.4247, Truncation lag parameter = 7, p-value = 0.01
```

```r
x <- 0.3*(1:1000)+rnorm(1000)  # is trend stationary
kpss.test(x, null = "Trend")
```

```
## Warning in kpss.test(x, null = "Trend"): p-value greater than printed p-
## value

##
##  KPSS Test for Trend Stationarity
##
## data:  x
## KPSS Trend = 0.034062, Truncation lag parameter = 7, p-value = 0.1
```

Apply to our transforms

Recap: We can more accurately check if the time series is stationary by using the Augmented Dickey-Fuller Test (adf test). *A p-Value of less than 0.05 in adf.test() indicates that it is stationary - alternative hypothesis: stationary is significant, reject null hypothesis.* KPSS test is used in complement to ADF. If the result from both tests suggests that the time series in stationary, then it probably is.

logdiff1

```r
# using our transforms
adf.test(values_logdiff1) # p-value < 0.05 indicates the TS is stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values_logdiff1
## Dickey-Fuller = -1.9169, Lag order = 3, p-value = 0.605
## alternative hypothesis: stationary
```

```r
kpss.test(values_logdiff1, null="Trend", lshort = FALSE) # trend/level stationarity test
```

```
##
##  KPSS Test for Trend Stationarity
##
## data:  values_logdiff1
## KPSS Trend = 0.13229, Truncation lag parameter = 9, p-value =
```

```
## 0.07539
```

ADF test shows the data after a log diff is not stationary.

logdiff2 (truncated)

```r
#values.logdiff.trunc
adf.test(values.logdiff.trunc)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  values.logdiff.trunc
## Dickey-Fuller = -2.7396, Lag order = 3, p-value = 0.2886
## alternative hypothesis: stationary
```

```r
kpss.test(values.logdiff.trunc)
```

```
## Warning in kpss.test(values.logdiff.trunc): p-value greater than printed p-
## value
```

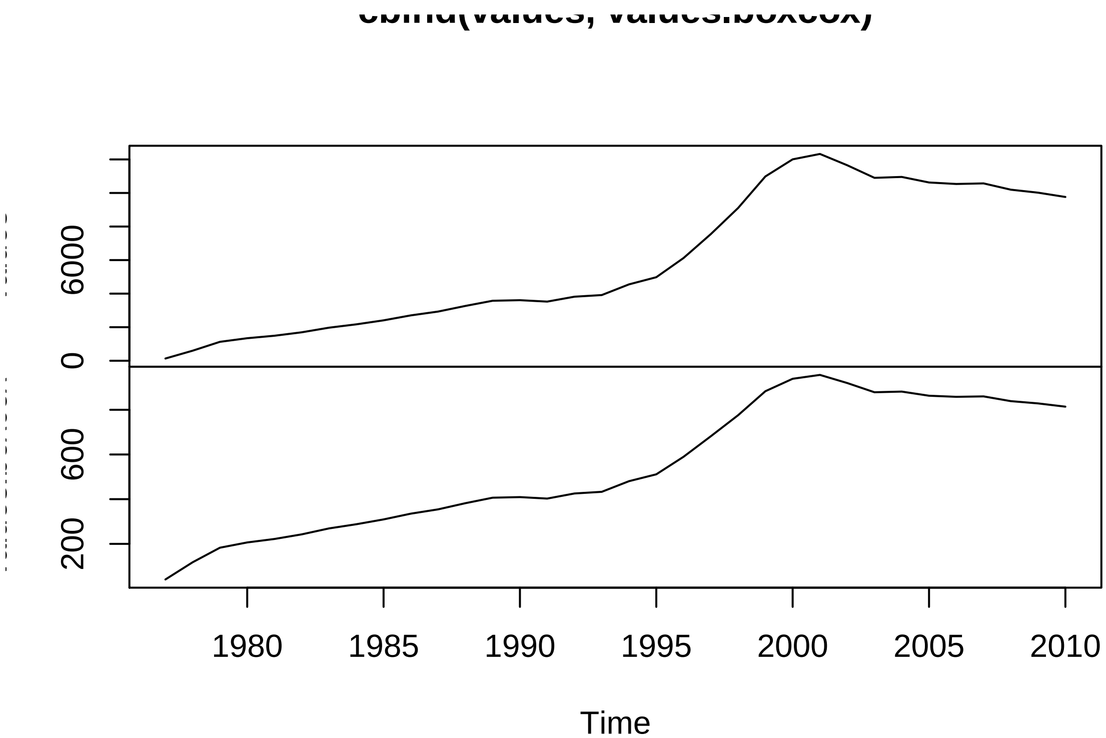
```
##
##  KPSS Test for Level Stationarity
##
## data:  values.logdiff.trunc
## KPSS Level = 0.06296, Truncation lag parameter = 2, p-value = 0.1
```

Still not getting stationarity with the truncated series and 2nd difference of log values… This data seems very hard to stationarize, so it might require us to fit a model that is not ARIMA, but instead some sort of decomposition, or smoothing, or regression model.

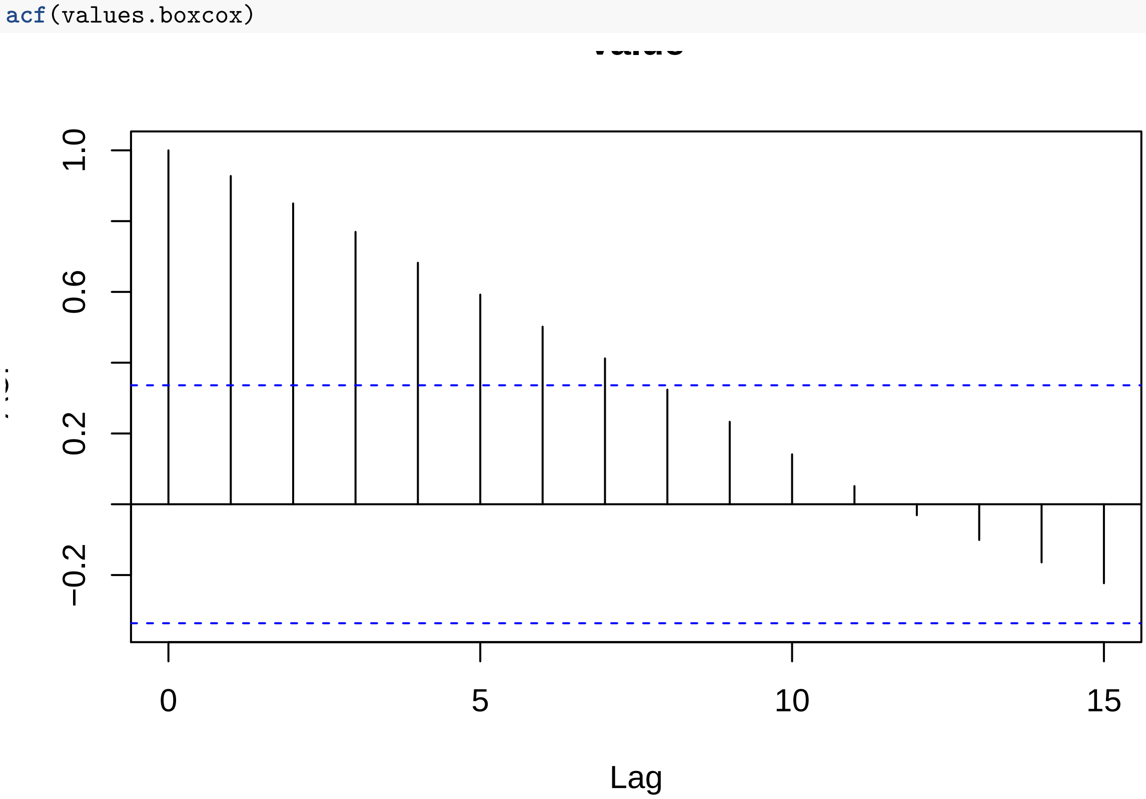
Box-Cox

TODO

```r
values.boxcox <- BoxCox(values, lambda="auto")
plot(cbind(values, values.boxcox))
```




```
acf(values.boxcox)
```



3. Decomposition

Decomposition of the series into trend and random components. (Can't use decompose() or stl() here because there is no seasonality component - fit a non parametric method instead to estimate a trend)

1. Trend estimation

Non-parametric: One approach is to estimate the trend with a smoothing procedure such as moving averages. (See Lesson 5.2 for more on that.) With this approach no equation is used to describe trend. Parametric: The second approach is to model the trend with a regression equation.

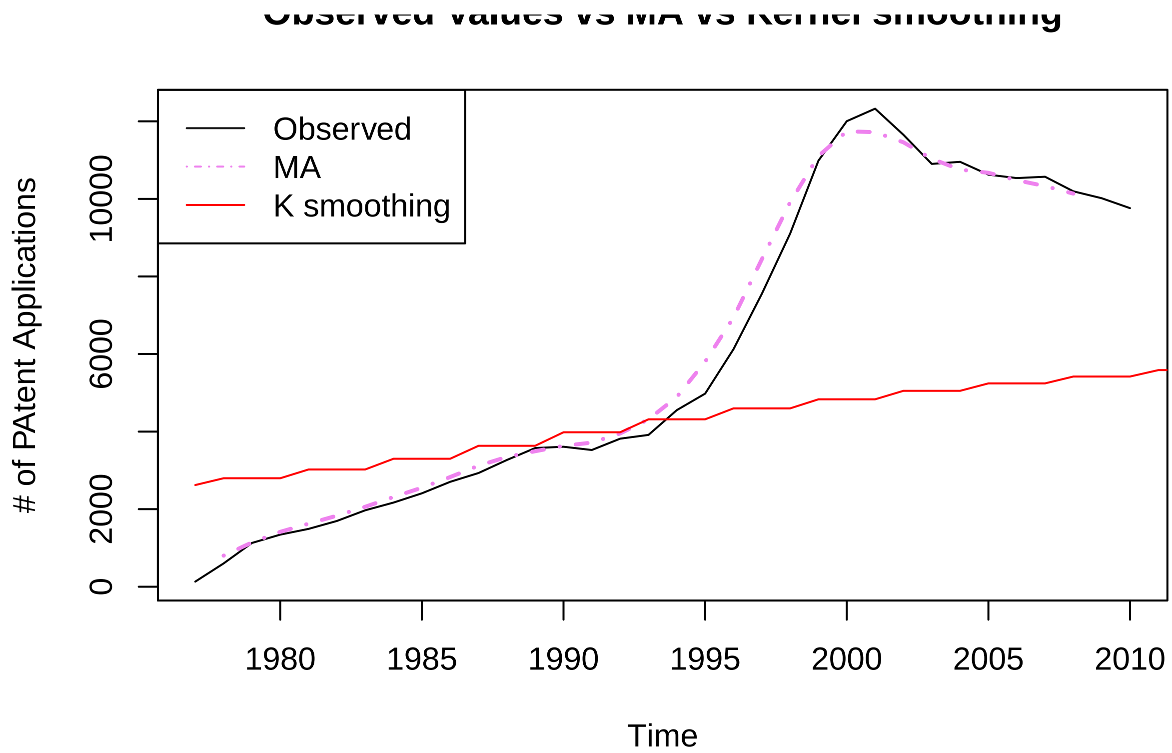
Smoothing

> To estimate the trend component of a non-seasonal time series that can be described using an additive model, it is common to use a smoothing method, such as calculating the simple moving average of the time series.
>
> The various exponential smoothing models are special cases of ARIMA models and can be fitted with ARIMA software. In particular, the simple exponential smoothing model is an ARIMA(0,1,1) model, Holt's linear smoothing model is an ARIMA(0,2,2) model, and the damped trend model is an ARIMA(1,1,2) model.
>
> Linear, quadratic, or exponential trend line models are other options for extrapolating a deseasonalized series, but they rarely outperform random walk, smoothing, or ARIMA models on business data.

Fit a moving average

```
#Create equally spaced time points for fitting trends
time.pts = c(1:length(values))
time.pts = c(time.pts - min(time.pts))/max(time.pts)
```

```
#using moving average method from forecast package
ma.values <- ma(values, order=4, centre = FALSE)

#define mav method and fit
mav.fit = ksmooth(time.pts, values, kernel = "box", bandwidth = 1.1)
values.fit.mav = ts(mav.fit$y,start=1977,frequency=1)

# plot mav.fit against values
ts.plot(values,ylab="# of PAtent Applications", main="Observed Values vs MA vs Kernel smoothing")
lines(ma.values,lwd=2, lty=4, col="violet")
lines(values.fit.mav, col="red")
legend(x="topleft", c("Observed", "MA", "K smoothing"), col=c("gray10","violet", "red"), lty=c(1, 4))
```




```
#values.fit.mav is the mav dataframe (type of the objects is float) with the transformed values for eac
#ablines is an a, b line graphing function, a is y intercept and b is slope
```

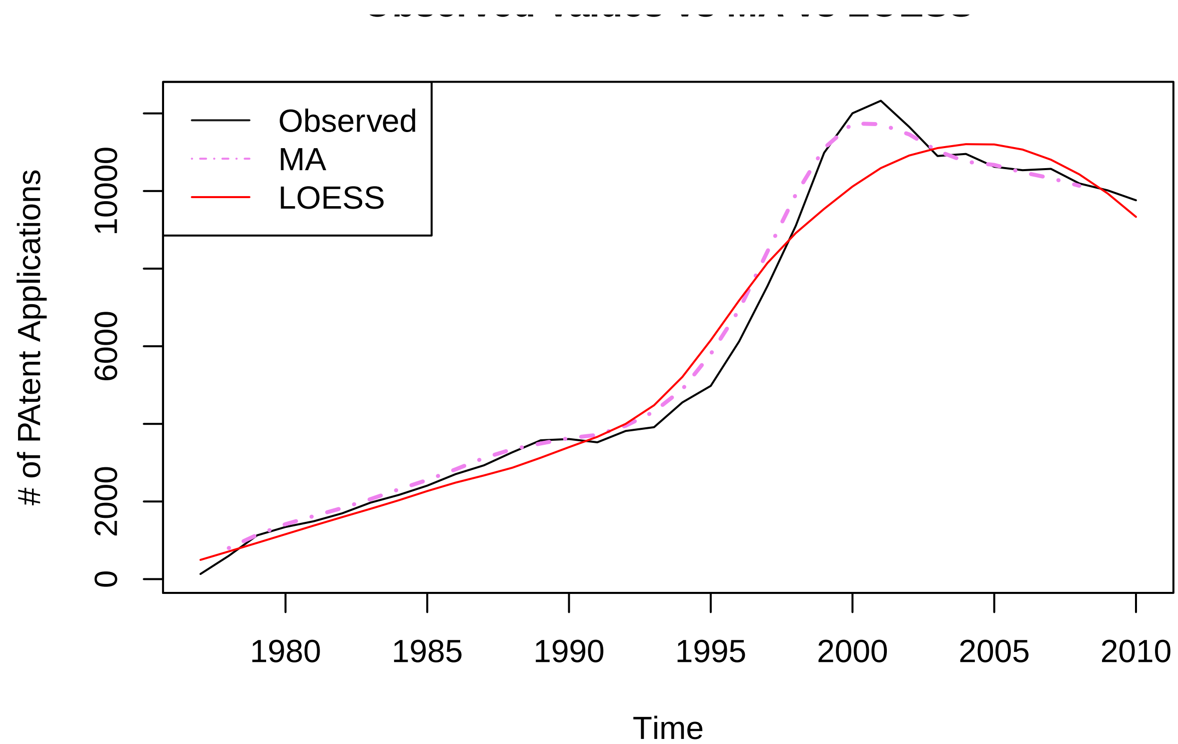
LOESS

locally estimated scatterplot smoothing

> Loess Regression is the most common method used to smoothen a volatile time series. It is a non-parametric methods where least squares regression is performed in localized subsets, which makes it a suitable candidate for smoothing any numerical vector.

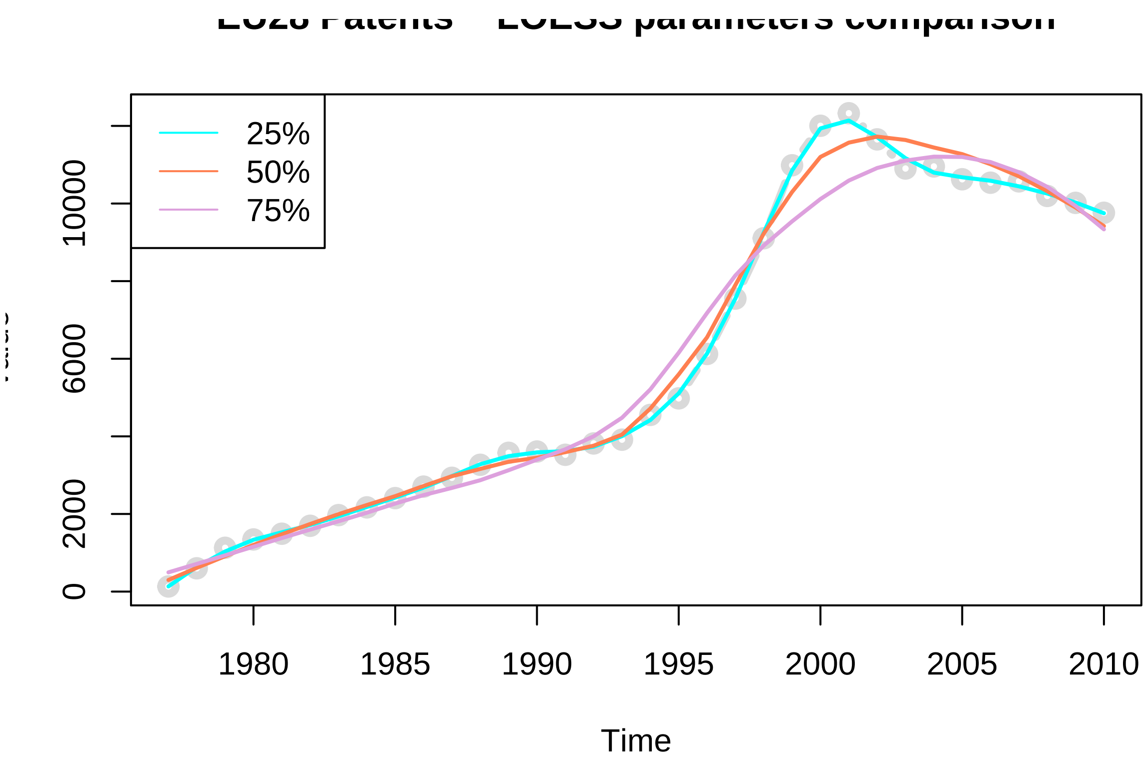
Span controls the degree of smoothing (greater values, smoother fit) We won't use predictor/explanatory variables, just the years of the observations.

```
loess.fit = loess(as.matrix(values)~time.pts, degree=2)
values.fit.loess = ts(fitted(loess.fit),start=1977)
#plot(values.fit.loess)

# plot LOESS against observ and ma
ts.plot(values,ylab="# of PAtent Applications", main="Observed Values vs MA vs LOESS")
lines(ma.values,lwd=2, lty=4, col="violet")
lines(values.fit.loess, col="red")
legend(x="topleft", c("Observed", "MA", "LOESS"), col=c("gray10","violet", "red"), lty=c(1, 4))
```



Tweaking span (default value is 0.75):

```
loess_fit25 <- loess(as.matrix(values)~time.pts, data=values, span=0.25) #25%smoothing span
loess_fit50 <- loess(as.matrix(values)~time.pts, data=values, span=0.5) #50%smoothing span

smoothed.values.loess25 <- ts(fitted(loess_fit25), start=1977)
smoothed.values.loess50 <- ts(predict(loess_fit50), start=1977)

plot(values, type="b", lwd=4, col="gray85", main="EU28 Patents - LOESS parameters comparison")
lines(smoothed.values.loess25, col="cyan", lwd=2, lty=1)
lines(smoothed.values.loess50, col="coral", lwd=2)
lines(values.fit.loess, col="plum", lwd=2)
legend(x="topleft", c("25%","50%","75%"),lty = 1, col=c("cyan","coral","plum"))
```



Finding optimal span

Find span that minimizes sum of squared errors - residuals

```
res.loess.75 <- sum(loess.fit$residuals^2)
res.loess.50 <- sum(loess_fit25$residuals^2)
res.loess.25 <- sum(loess_fit50$residuals^2)

sprintf("Residuals (SSE) at span 0.75: %f", res.loess.75)
```

```
## [1] "Residuals (SSE) at span 0.75: 14737870.042897"
```

```
sprintf("Residuals (SSE) at span 0.50: %f", res.loess.50)
```

```
## [1] "Residuals (SSE) at span 0.50: 287856.406508"
```

```
sprintf("Residuals (SSE) at span 0.25: %f", res.loess.25)
```

```
## [1] "Residuals (SSE) at span 0.25: 4232049.516755"
```

"Residuals (SSE) at span 0.75: 14737870.042897" "Residuals (SSE) at span 0.50: 287856.406508" "Residuals (SSE) at span 0.25: 4232049.516755"

Loess at 50% span seems like a better fit than 0.25 or 0.75

There are also some optimizer functions to find the minimum or maximum parameters, for instance: (Code adapted from http://r-statistics.co/Loess-Regression-With-R.html)

```
# define function that returns the SSE
calcSSE <- function(x, dts=values){
  sse =0
  print("hello")
  loessMod <- try(loess(as.matrix(dts) ~ time.pts, data=dts, span=x), silent=T)
  print("here")
```

```r
  res <- try(loessMod$residuals, silent=T)
  if(class(res)!="try-error"){
    if((sum(abs(res), na.rm=T) > 0)){
      sse <- sum(res^2)
      print("over here")
    }

  } else{
    sse <- 99999
    print("no else")
  }
  return(sse)
}

# Run optimizing function to find span that gives min span, starting at 0.155
#optim(par=c(0.2), calcSSE, method="SANN") #this takes some time to run
optimize(calcSSE, c(0.1, 0.75))
```

```
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"
```

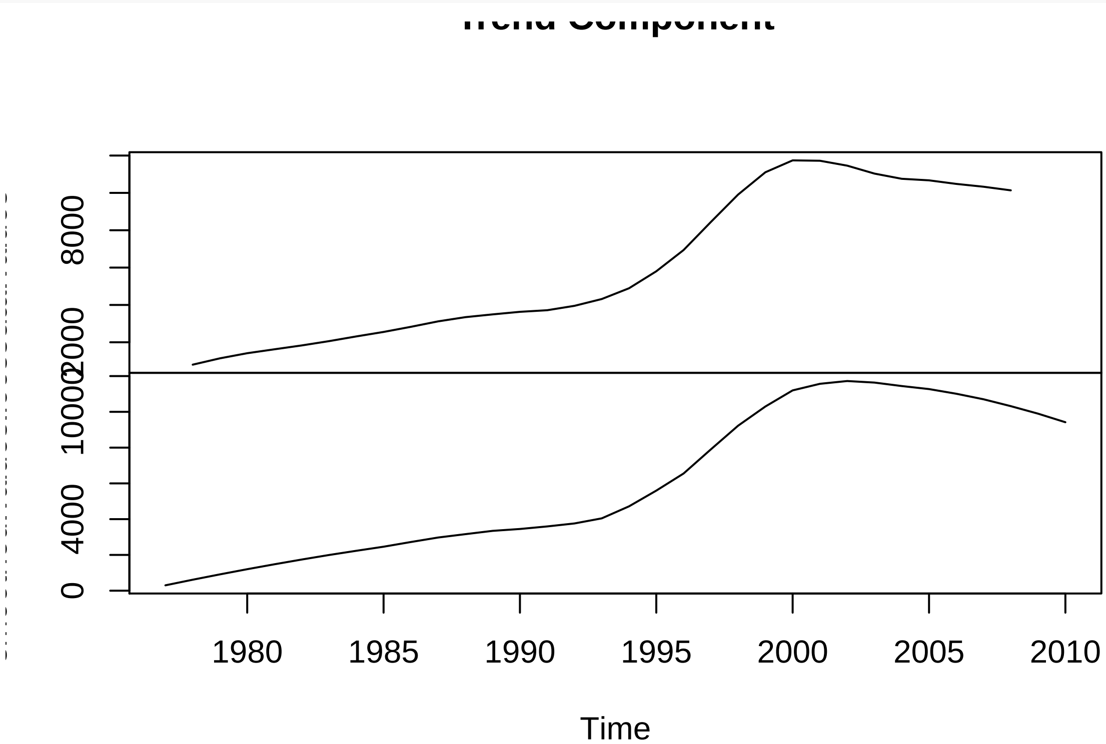
```
## [1] "hello"
## [1] "here"
## [1] "over here"
## [1] "hello"
## [1] "here"
## [1] "over here"

## $minimum
## [1] 0.2535181
##
## $objective
## [1] 287856.4
```

Detrend

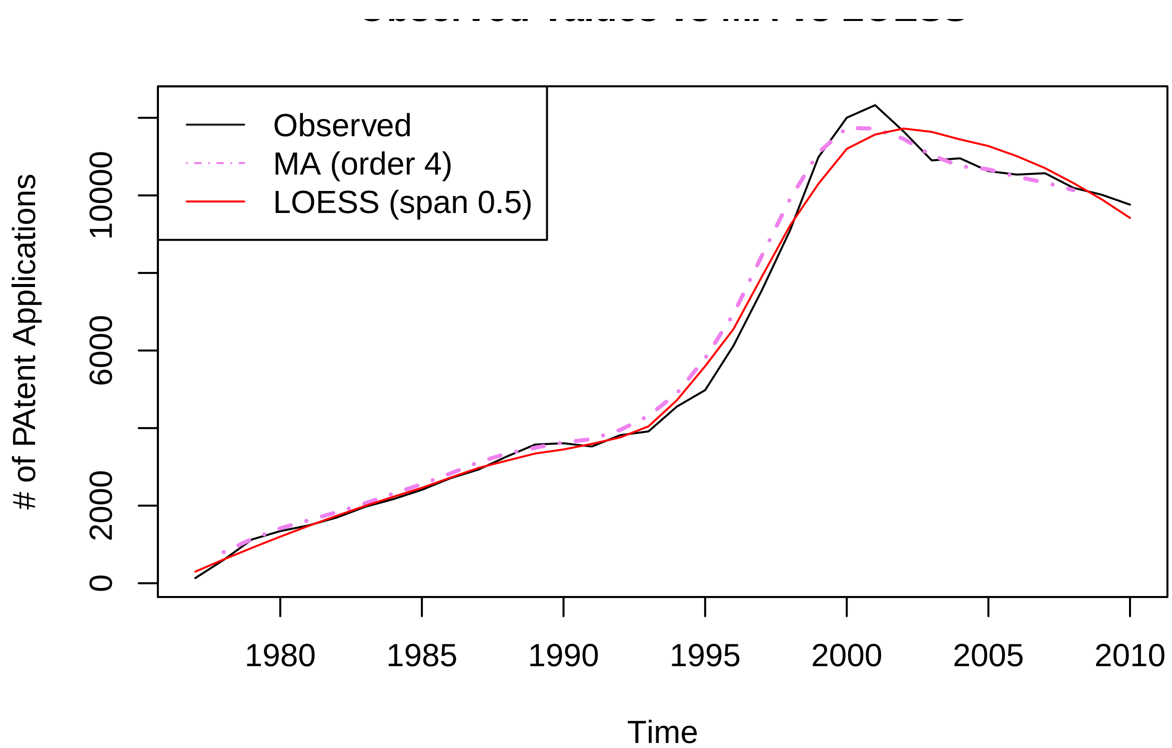
2. "De-trend" the series. For an additive decomposition, this is done by subtracting the trend estimates from the series. For a multiplicative decomposition, this is done by dividing the series by the trend values.

Using MA and LOESS at 0.5 span

```r
plot(cbind(ma.values, smoothed.values.loess50), main = "Trend Component")
```

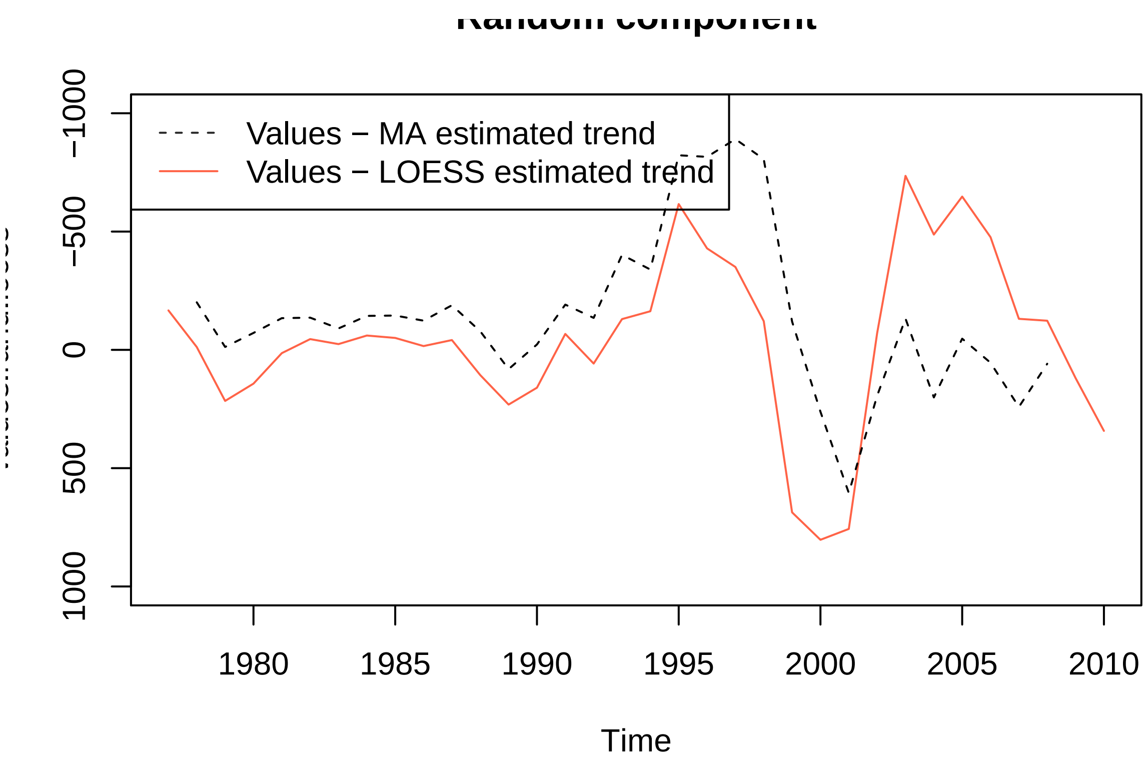



```r
# plot LOESS against observ and ma
ts.plot(values,ylab="# of PAtent Applications", main="Observed Values vs MA vs LOESS")
lines(ma.values,lwd=2, lty=4, col="violet")
lines(smoothed.values.loess50, col="red")
legend(x="topleft", c("Observed", "MA (order 4)", "LOESS (span 0.5)"), col=c("gray10","violet", "red"),
```



##Random component 3. The final step is to determine the random (irregular) component.

```
values.rand.ma <- values-ma.values

values.rand.loess <- ts(loess_fit50$residuals, start = 1977)

plot(values.rand.loess, col="tomato", pty="l", main="Random component", ylim=c(1000, -1000))
lines(values.rand.ma, lty=2)
legend(x="topleft", legend=c("Values - MA estimated trend", "Values - LOESS estimated trend"),
       col=c("gray15", "tomato"), lty=c(2, 1))
```



The random component could be analyzed for such things as the mean location, or mean squared size (variance), or possibly even for whether the component is actually random or might be modeled with an ARIMA model.

---

4. Model Selection

Exponential Smoothing (ETS)

For a time series that can be described using an additive model with constant level and no seasonality, we can use simple exponential smoothing to make short-term forecasts. (we can use our differenced series, but Holt's smoothing as we see next makes more sense here)

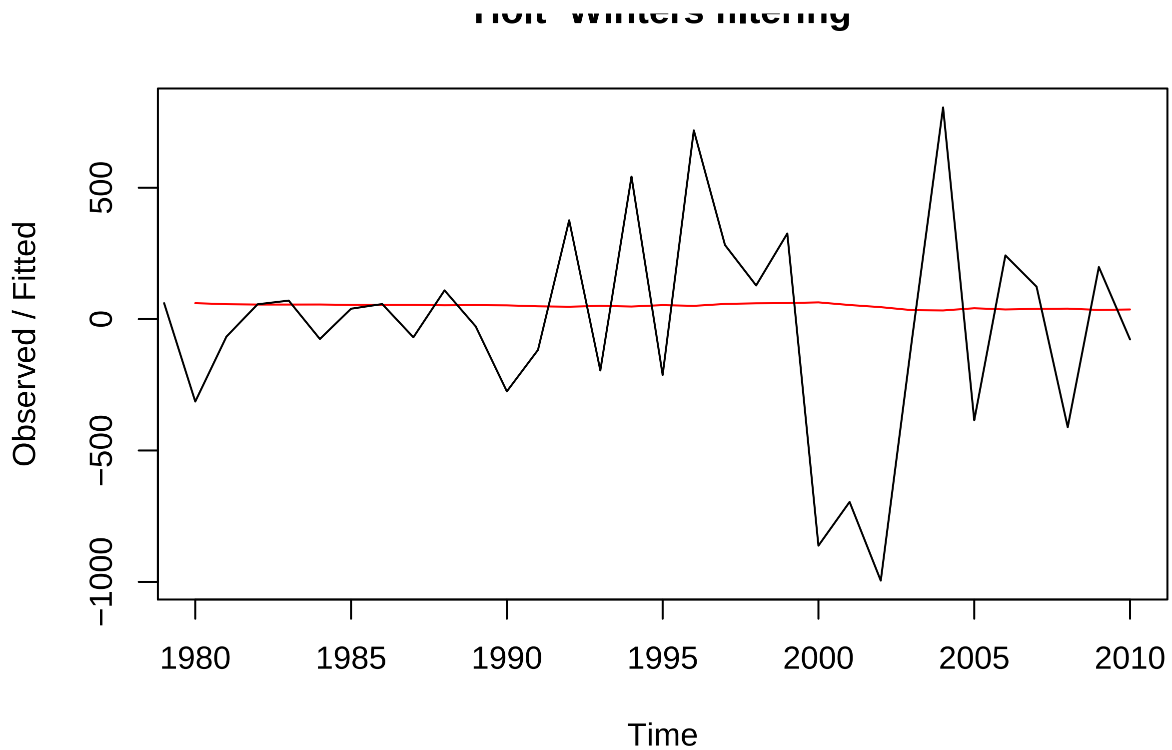
Holt-Winters

```
values.hw <- HoltWinters(values_diff2, beta=FALSE, gamma=FALSE)
values.hw
```

```
## Holt-Winters exponential smoothing without trend and without seasonal component.
##
## Call:
## HoltWinters(x = values_diff2, beta = FALSE, gamma = FALSE)
##
## Smoothing parameters:
##  alpha: 0.01088974
##  beta : FALSE
##  gamma: FALSE
##
## Coefficients:
##       [,1]
```

```
## a 35.46834
```

```r
#fitted(values.hw)
plot(values.hw)
```




```r
values.hw$SSE
```

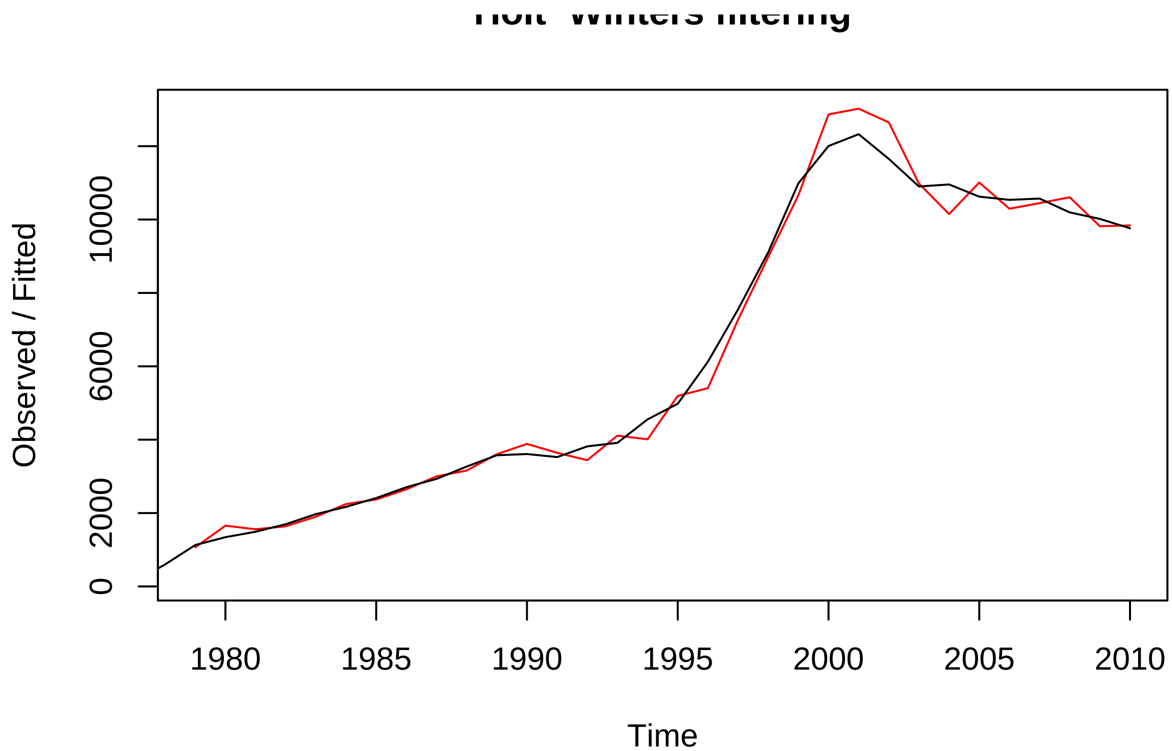
```
## [1] 4974147
```

Holt's exponenetial smoothing is a better fit for the characteristic of this series, as it contemplates a trend componenet and no seasonality.

```r
values.hsmooth <- HoltWinters(values, gamma=FALSE)
values.hsmooth
```

```
## Holt-Winters exponential smoothing with trend and without seasonal component.
##
## Call:
## HoltWinters(x = values, gamma = FALSE)
##
## Smoothing parameters:
##  alpha: 1
##  beta : 1
##  gamma: FALSE
##
## Coefficients:
##      [,1]
## a 9761.64
## b -255.40
```

```r
#fitted(values.hw)
plot(values.hsmooth)
```



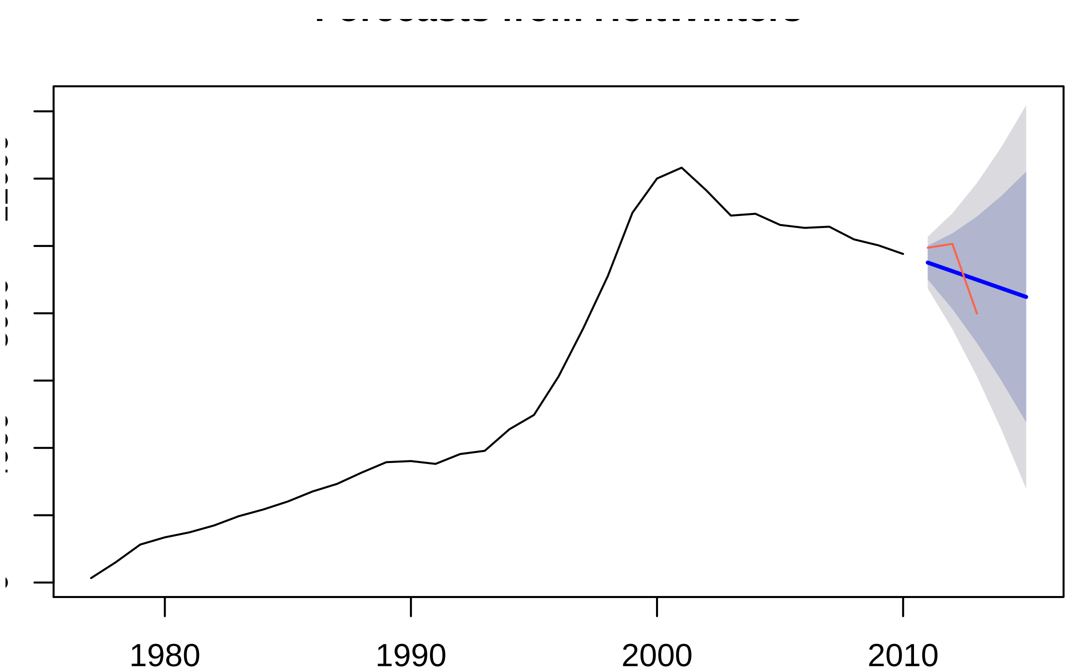

```r
values.hsmooth$SSE
```

```
## [1] 4778807
```

The estimated value of alpha and beta is 1.00 - "both are high, telling us that both the estimate of the current value of the level, and of the slope b of the trend component, are based mostly upon very recent observations in the time series. This makes good intuitive sense, since the level and the slope of the time series both change quite a lot over time." (https://a-little-book-of-r-for-time-series.readthedocs.io/en/latest/src/timeseries.html#decomposing-non-seasonal-data)

The value of the sum-of-squared-errors for the in-sample forecast errors is 4,778,807.

```r
forecast.values.hwsmooth <- forecast(values.hsmooth, h=5) #forecast next 5 years
plot(forecast.values.hwsmooth)
lines(values.holdout, col="tomato")
```



ETS(A,A,N):

Holt's linear method with additive errors

Parameters for exponential smoothing can also be estimated automatically by using:

for AAN

```
values.fit.AANets <- ets(values, model="AAN")
coef(values.fit.AANets)
```

```
##       alpha        beta         phi           l           b
##   0.9998999   0.9998986   0.8137758 127.8187459 287.3699271
```

```
summary(values.fit.AANets)
```

```
## ETS(A,Ad,N)
##
## Call:
##  ets(y = values, model = "AAN")
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.9999
##     phi   = 0.8138
##
##   Initial states:
##     l = 127.8187
##     b = 287.3699
##
##   sigma:  398.2461
##
##      AIC     AICc      BIC
## 533.6088 536.7199 542.7670
##
## Training set error measures:
##                    ME     RMSE      MAE      MPE     MAPE      MASE
```

```
## Training set 39.77987 367.7995 282.2032 -1.76782 12.05527 0.6165582
##                      ACF1
## Training set 0.1091697
```

for MAN

```
values.fit.MANets <- ets(values, model="MAN")
coef(values.fit.MANets)
```

```
##        alpha         beta            l            b
##    0.9713697    0.8873032 -323.2920999  453.6989919
```

```
summary(values.fit.MANets)
```

```
## ETS(M,A,N)
##
## Call:
##  ets(y = values, model = "MAN")
##
##   Smoothing parameters:
##     alpha = 0.9714
##     beta  = 0.8873
##
##   Initial states:
##     l = -323.2921
##     b = 453.699
##
##   sigma:  0.0657
##
##      AIC     AICc      BIC
## 507.6724 509.8153 515.3042
##
## Training set error measures:
##                       ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -23.1999 381.9091 265.6755 -0.2811852 4.762199 0.5804484
##                      ACF1
## Training set 0.2136499
```

Auto selection of ETS model

```
values.fit.autoets <- ets(values, model="ZZZ")
coef(values.fit.autoets)
```

```
##        alpha         beta            l            b
##    0.9713697    0.8873032 -323.2920999  453.6989919
```

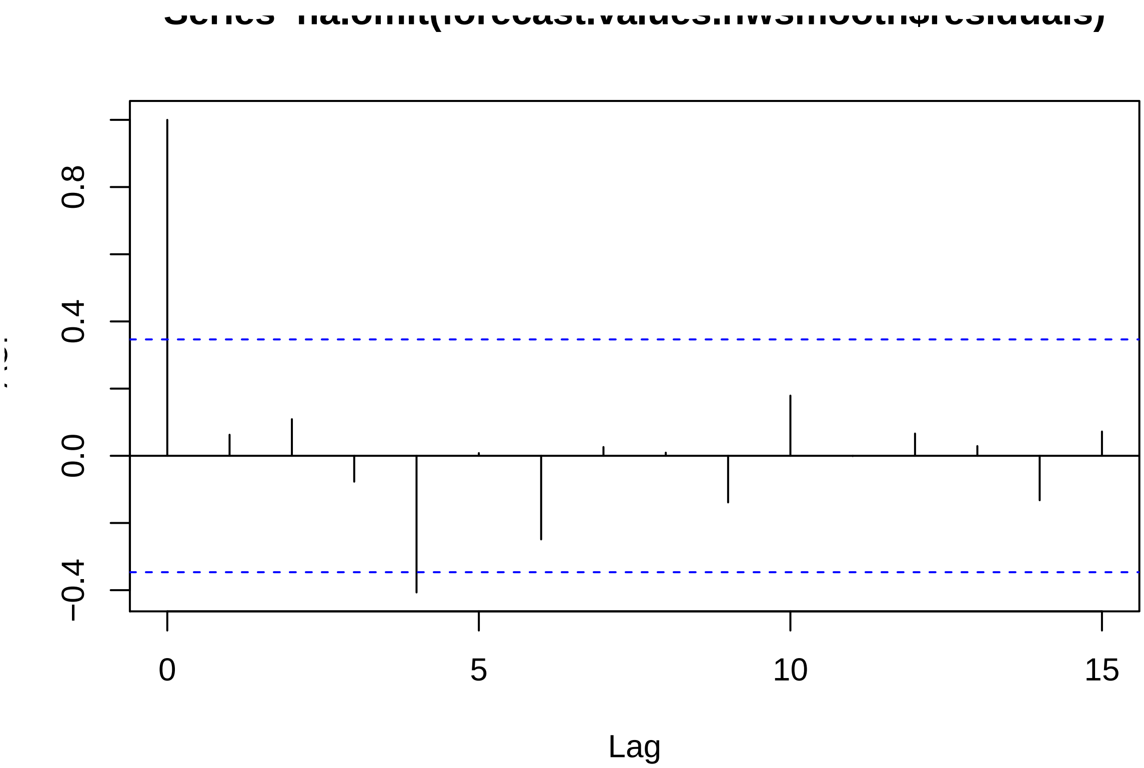
```
summary(values.fit.autoets)
```

```
## ETS(M,A,N)
##
## Call:
##  ets(y = values, model = "ZZZ")
##
##   Smoothing parameters:
##     alpha = 0.9714
##     beta  = 0.8873
##
##   Initial states:
```

```
##     l = -323.2921
##     b = 453.699
##
##   sigma:  0.0657
##
##      AIC     AICc      BIC
## 507.6724 509.8153 515.3042
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -23.1999 381.9091 265.6755 -0.2811852 4.762199 0.5804484
##                    ACF1
## Training set 0.2136499
```

Residuals of holt's smoothing

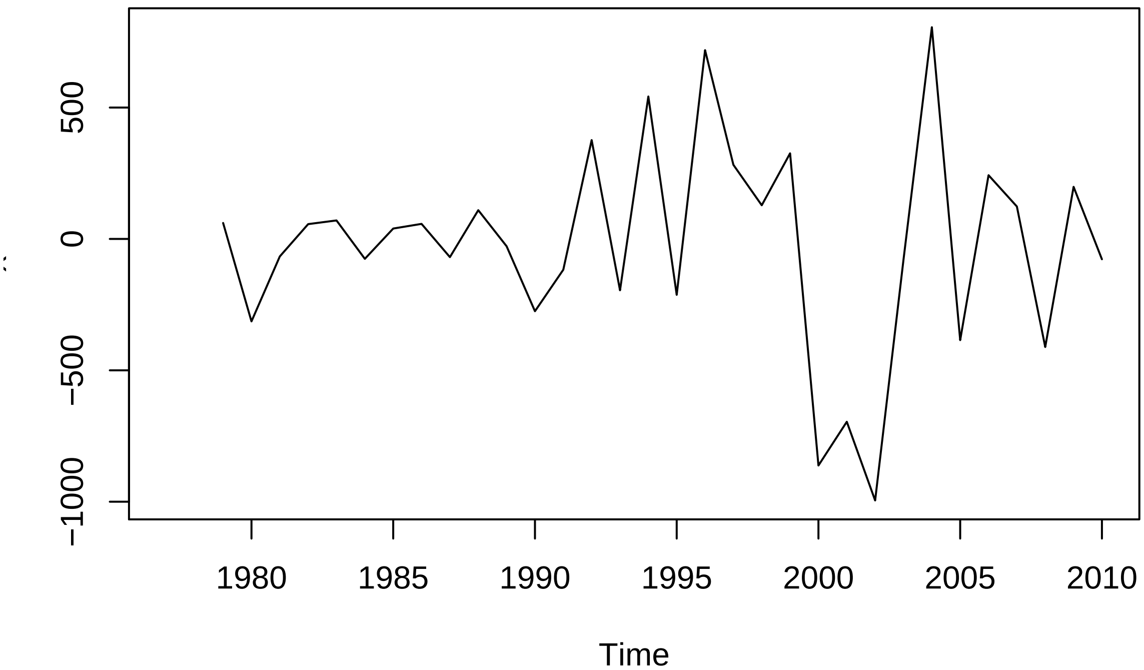
```r
acf(na.omit(forecast.values.hwsmooth$residuals))
```




```r
Box.test(forecast.values.hwsmooth$residuals, type="Ljung-Box")
```

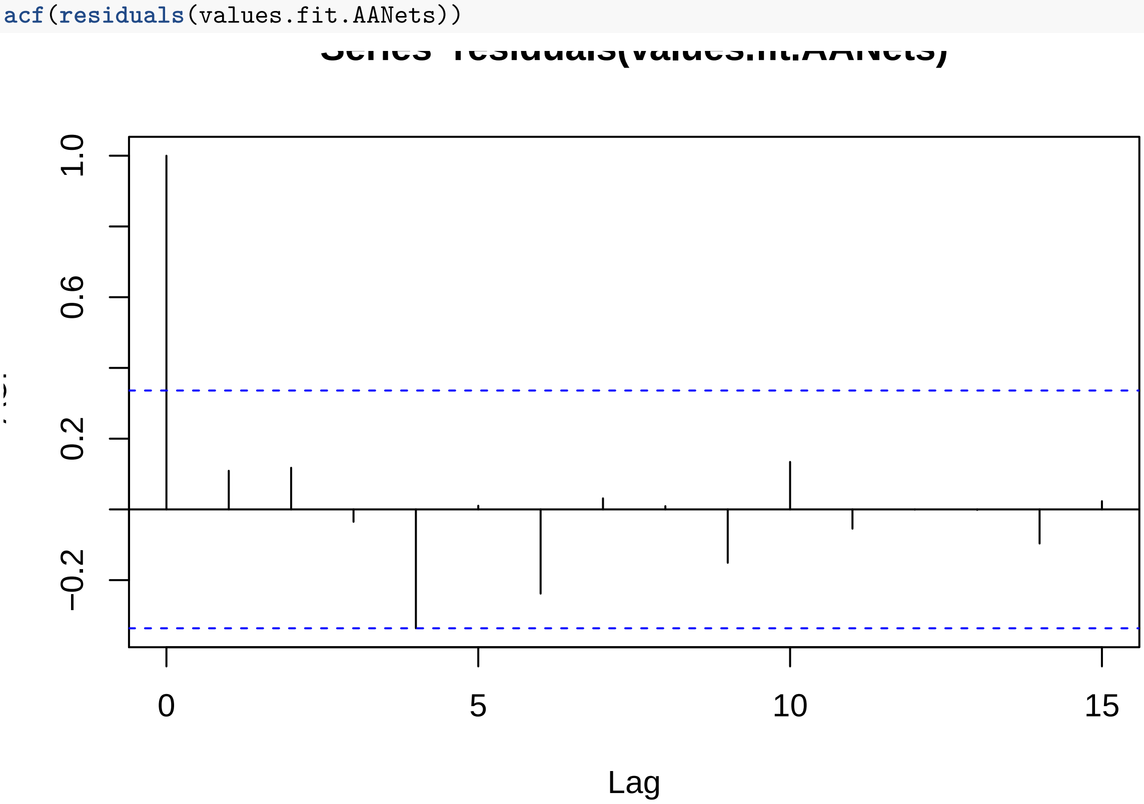
```
##
##  Box-Ljung test
##
## data:  forecast.values.hwsmooth$residuals
## X-squared = 0.13785, df = 1, p-value = 0.7104
```

```r
plot(forecast.values.hwsmooth$residuals)
```



Residuals of AAN ETS - trend dampened

```
acf(residuals(values.fit.AANets))
```




```
Box.test(residuals(values.fit.AANets), type="Ljung-Box")
```

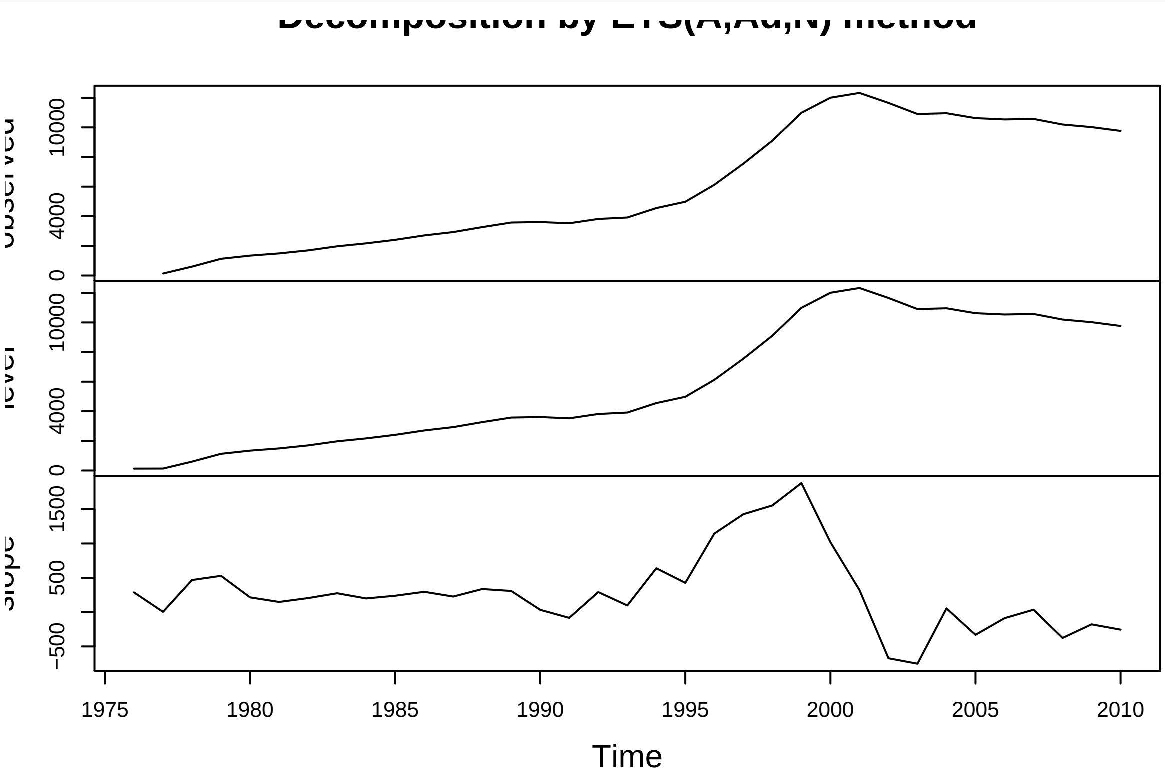
```
##
##  Box-Ljung test
```

```
## 
## data:  residuals(values.fit.AANets)
## X-squared = 0.44205, df = 1, p-value = 0.5061
```

```r
accuracy(values.fit.AANets)
```

```
##                    ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 39.77987 367.7995 282.2032 -1.76782 12.05527 0.6165582
##                   ACF1
## Training set 0.1091697
```

```r
plot(values.fit.AANets)
```

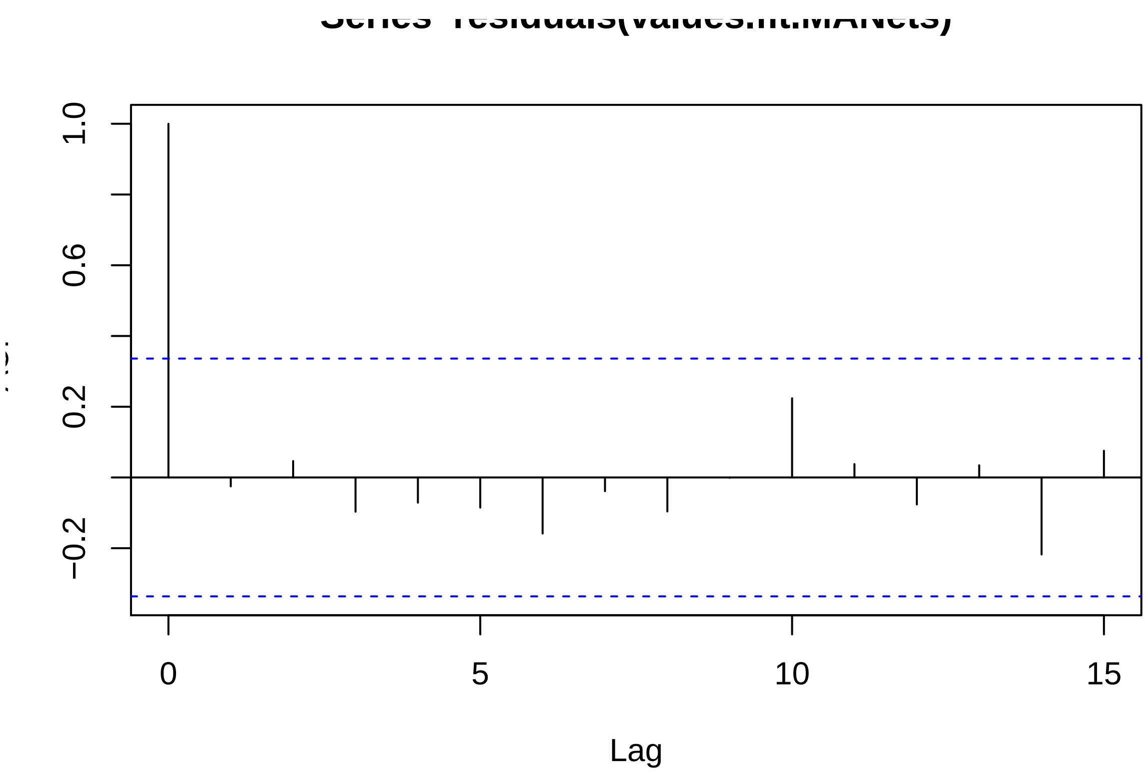



```r
#plot(forecast(values.fit.autoets, h=5))
```

no residuals autocorrelation MAPE at 12.06%, RMSE 367.8

Residuals of ETS(MAN) - holt's with multiplicative errors

```r
acf(residuals(values.fit.MANets))
```



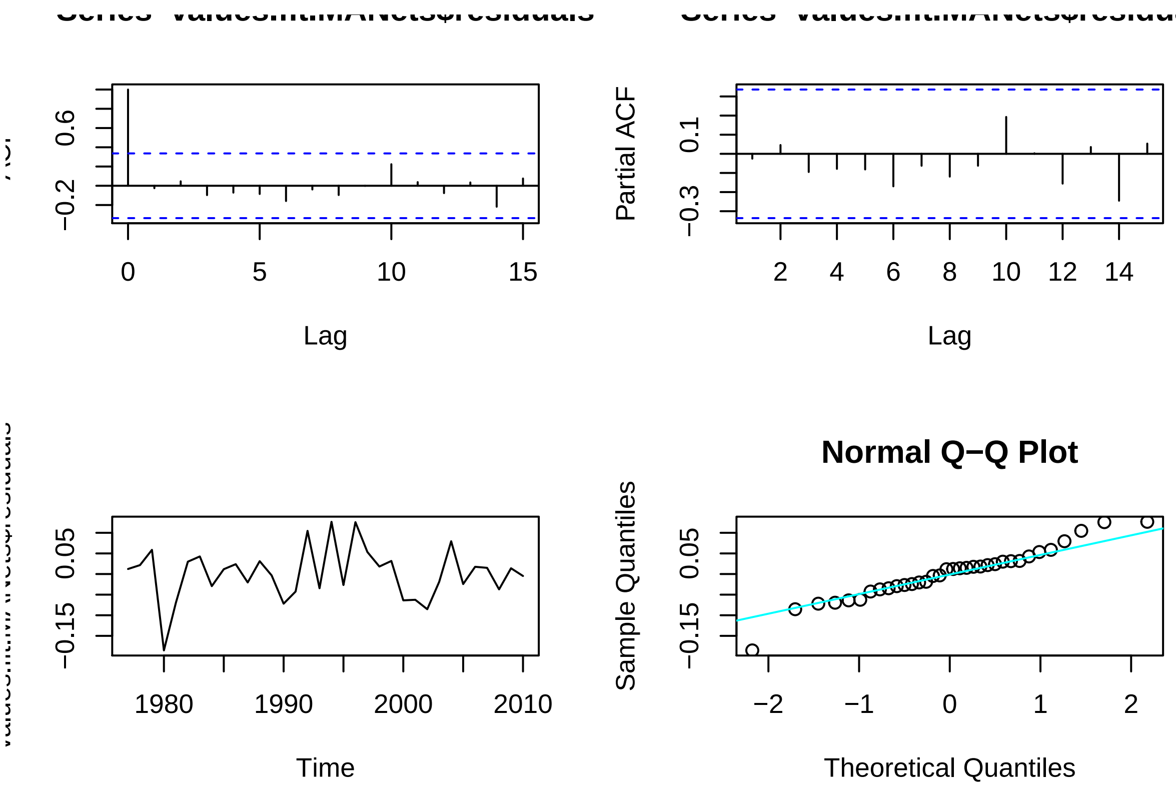

```
Box.test(residuals(values.fit.MANets), type="Ljung-Box") #test alternative hypothesis that there is non
```

```
##
##  Box-Ljung test
##
## data:  residuals(values.fit.MANets)
## X-squared = 0.023691, df = 1, p-value = 0.8777
```

```
accuracy(values.fit.MANets)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -23.1999 381.9091 265.6755 -0.2811852 4.762199 0.5804484
##                    ACF1
## Training set 0.2136499
```

```
par(mfrow=c(2,2))
acf(values.fit.MANets$residuals)
pacf(values.fit.MANets$residuals)
plot(values.fit.MANets$residuals)
qqnorm(values.fit.MANets$residuals)
qqline(values.fit.MANets$residuals, col="cyan")
```



good acf plot, confirmed by ljung-box (no autocorrelation in residuals) accuracy:mean absolute percent errors 4.76%, root mean sqr error 381.91

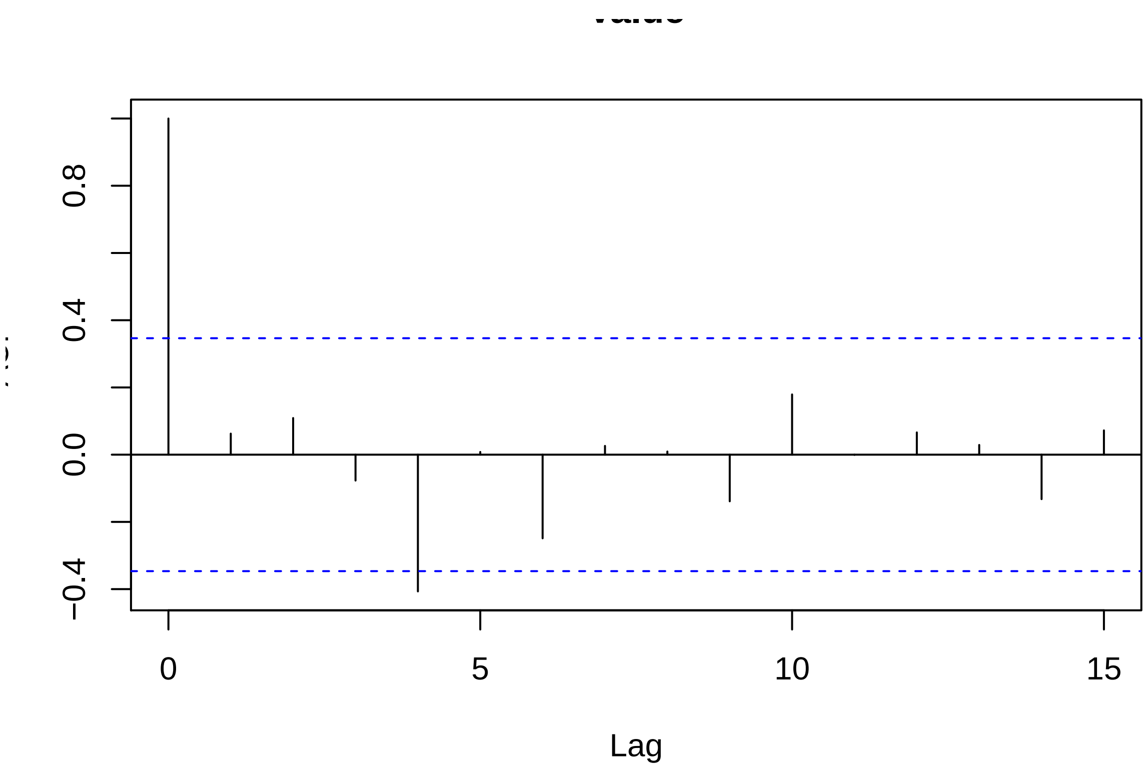
ARIMA

If you do not choose seasonal adjustment (or if the data are non-seasonal), you may wish to use the ARIMA model framework. ARIMA models are a very general class of models that includes random walk, random trend, exponential smoothing, and autoregressive models as special cases. ***The conventional wisdom is that a series is a good candidate for an ARIMA model if (i) it can be stationarized by a combination of differencing and other mathematical transformations such as logging, and (ii) you have a substantial amount of data to work with: at least 4 full seasons in the case of seasonal data.*** (If the series cannot be adequately stationarized by differencing–e.g., if it is very irregular or seems to be qualitatively changing its behavior over time–or if you have fewer than 4 seasons of data, then you might be better off with a model that uses seasonal adjustment and some kind of simple averaging or smoothing.)

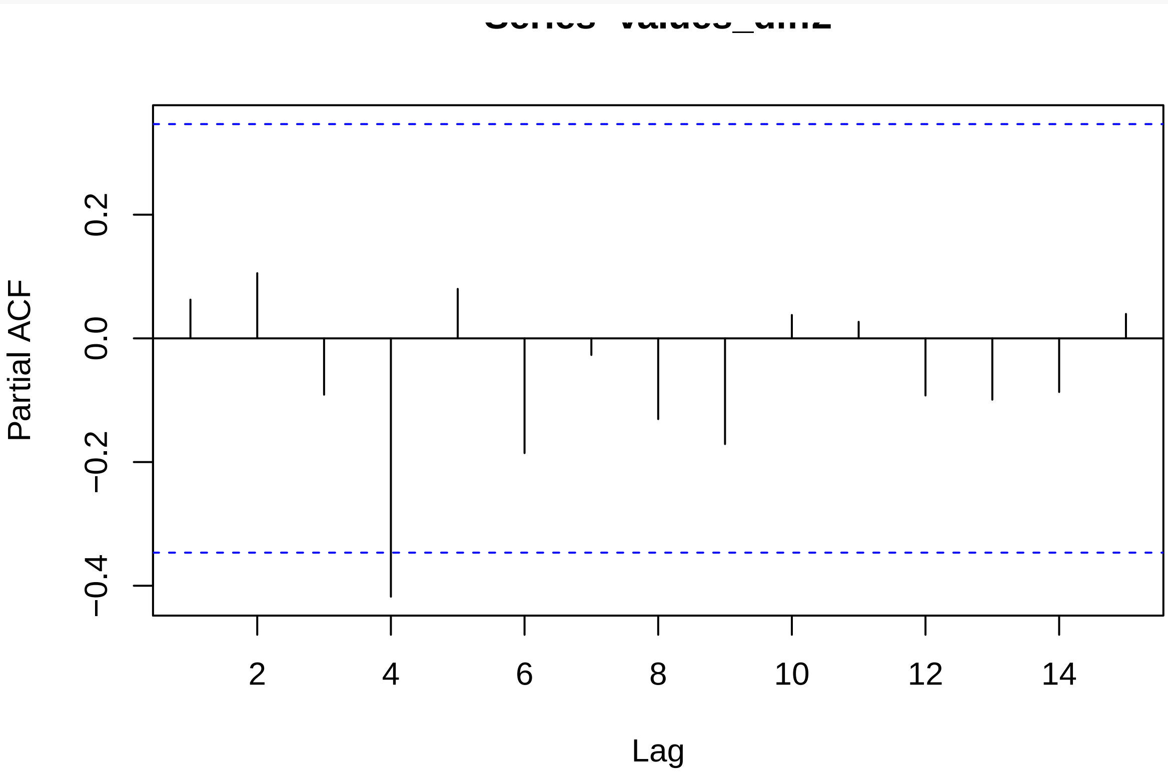
ARIMA(0, d, q)

We already established we need the second difference to stationarize this series, so d=2. looking at the ACF and PACF:

```
acf(values_diff2)
```




```r
pacf(values_diff2)
```



Auto arima

```r
auto.arima(values)
```

```
## Series: values
## ARIMA(1,1,0)
##
## Coefficients:
##          ar1
##       0.8084
## s.e.  0.0938
##
## sigma^2 estimated as 137743:  log likelihood=-242.09
## AIC=488.19   AICc=488.59   BIC=491.18
```

Auto ARIMA suggests only 1 difference, and looking at acf and pacf for the values after 1st difference, there seems to emerge an AR pattern (ACF decaying more slowly in a pattern, PACF spikes at lag 1). So taking the 2nd difference is probably overdifferencing

```r
kpss.test(values_diff1)
```

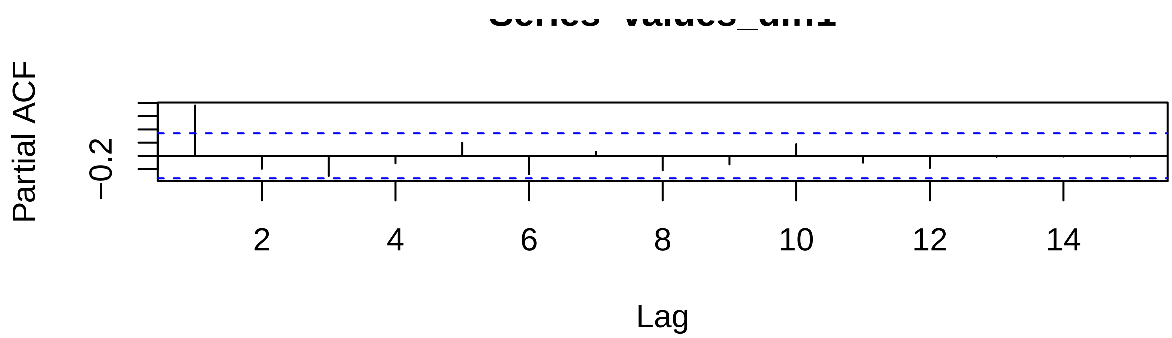
```
## Warning in kpss.test(values_diff1): p-value greater than printed p-value
```

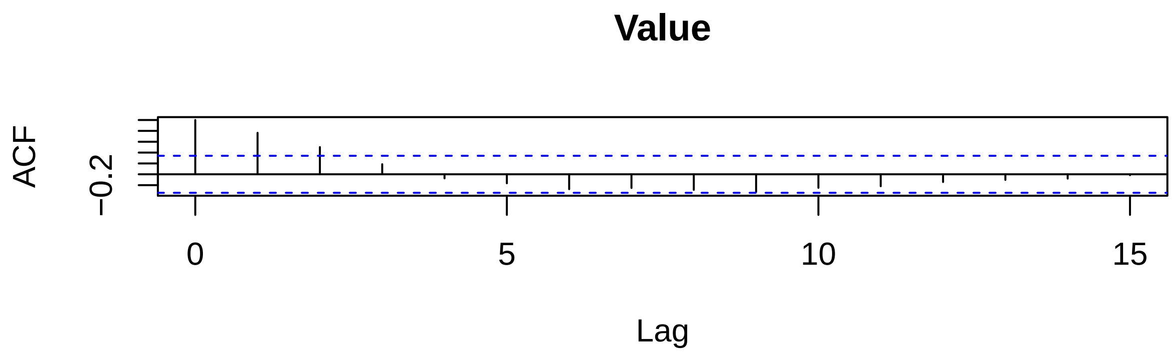
```
##
##  KPSS Test for Level Stationarity
##
## data:  values_diff1
## KPSS Level = 0.13964, Truncation lag parameter = 3, p-value = 0.1
```

```r
pp.test(values_diff1)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  values_diff1
## Dickey-Fuller Z(alpha) = -9.6062, Truncation lag parameter = 3,
## p-value = 0.513
## alternative hypothesis: stationary
```

```r
par(mfrow=c(2,1))
pacf(values_diff1)
acf(values_diff1)
```






```
values.fit.arima1_1_0 <- auto.arima(values)
values.fit.arima1_2_0 <- arima(values, order=c(1,2,0))
values.fit.arima0_2_1 <- arima(values, order=c(0,2,1))



summary(values.fit.arima1_1_0)
```

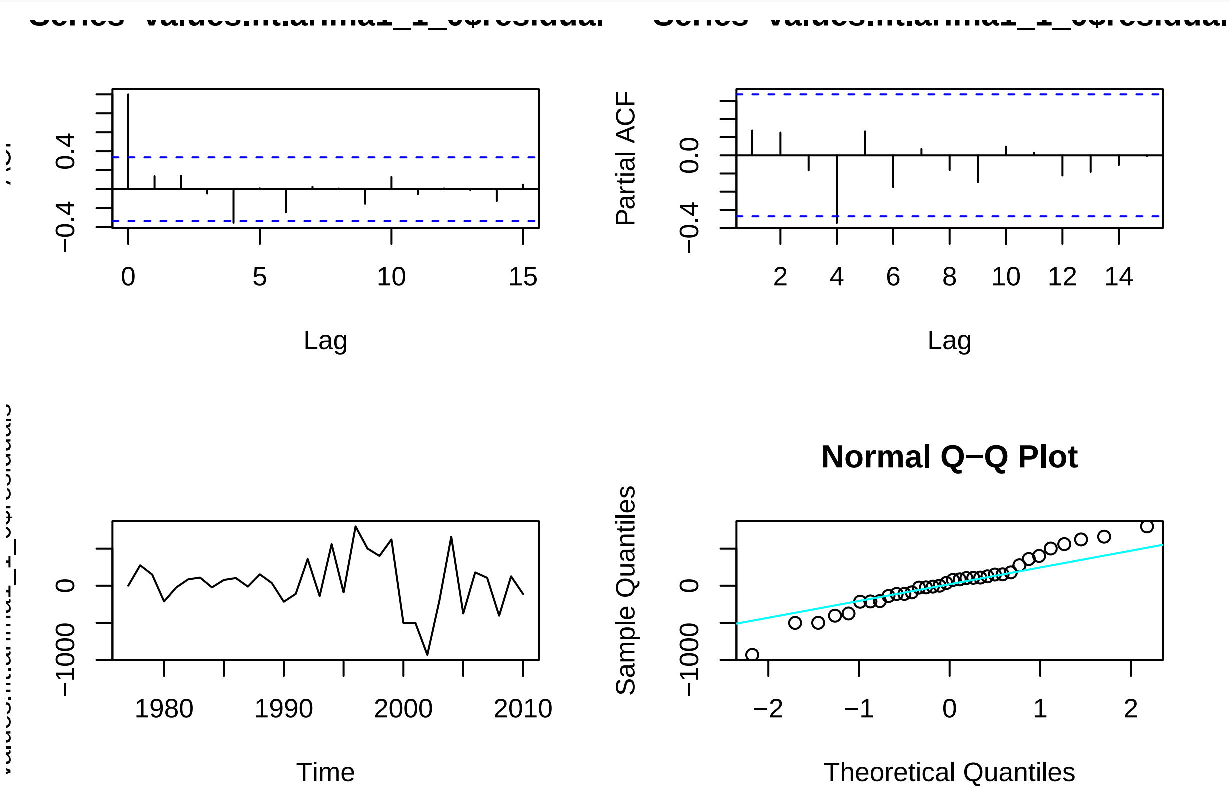
```
## Series: values
## ARIMA(1,1,0)
##
## Coefficients:
##          ar1
##       0.8084
## s.e.  0.0938
##
## sigma^2 estimated as 137743:  log likelihood=-242.09
## AIC=488.19   AICc=488.59   BIC=491.18
##
## Training set error measures:
##                    ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 42.55161 360.0566 269.8589 2.466738 6.014659 0.5895884
##                  ACF1
## Training set 0.137001
```

```
accuracy(values.fit.arima1_1_0)
```

```
##                    ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 42.55161 360.0566 269.8589 2.466738 6.014659 0.5895884
##                  ACF1
```

```
## Training set 0.137001
```

```r
#arima 110 residuals analysis
par(mfrow=c(2, 2))
acf(values.fit.arima1_1_0$residuals)
pacf(values.fit.arima1_1_0$residuals)
plot(values.fit.arima1_1_0$residuals)
qqnorm(values.fit.arima1_1_0$residuals)
qqline(values.fit.arima1_1_0$residuals, col="cyan")
```



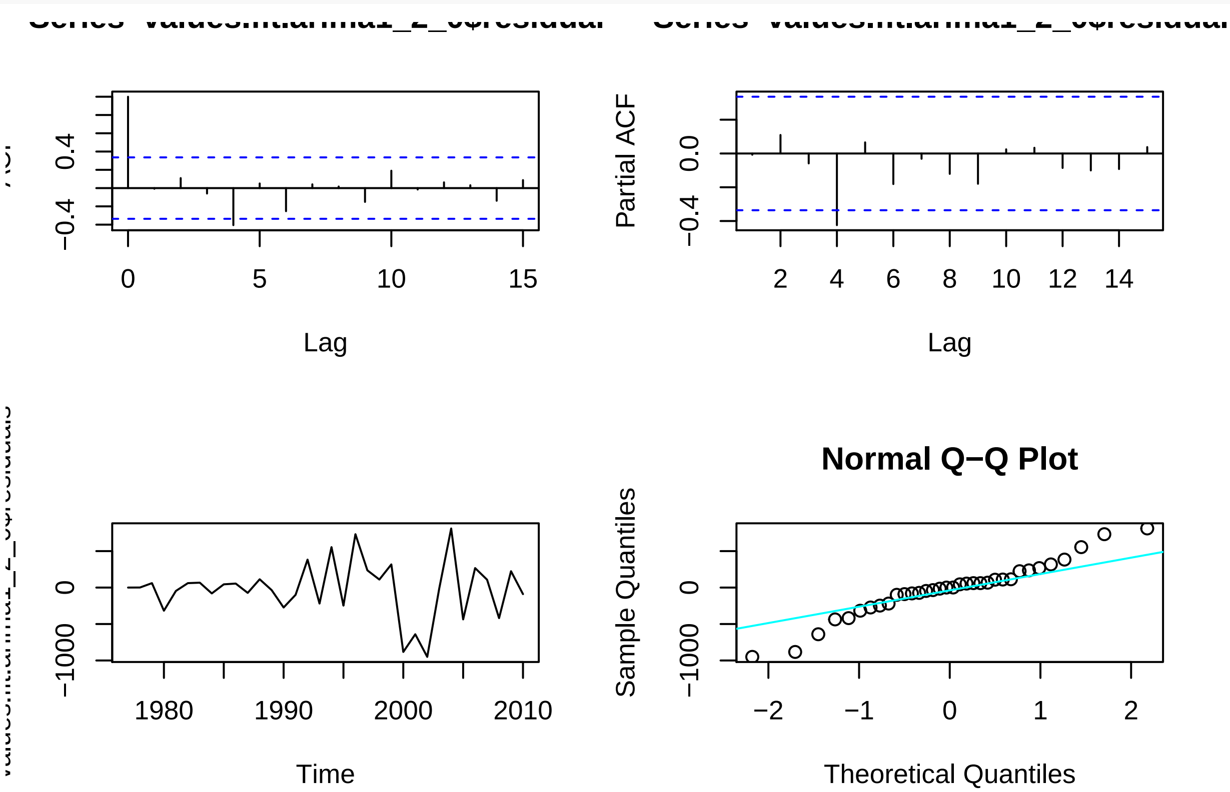

```r
summary(values.fit.arima1_2_0)
```

```
##
## Call:
## arima(x = values, order = c(1, 2, 0))
##
## Coefficients:
##          ar1
##       0.0640
## s.e.  0.1738
##
## sigma^2 estimated as 148688:  log likelihood = -235.96,  aic = 475.92
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -20.03002 374.0882 263.9179 -0.3062419 4.687829 0.5766085
##                      ACF1
## Training set -0.007508618
```

```r
accuracy(values.fit.arima1_2_0)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -20.03002 374.0882 263.9179 -0.3062419 4.687829 0.5766085
##                      ACF1
## Training set -0.007508618
```

```r
#arima 120 residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima1_2_0$residuals)
pacf(values.fit.arima1_2_0$residuals)
plot(values.fit.arima1_2_0$residuals)
qqnorm(values.fit.arima1_2_0$residuals)
qqline(values.fit.arima1_2_0$residuals, col="cyan")
```




```r
summary(values.fit.arima0_2_1)
```

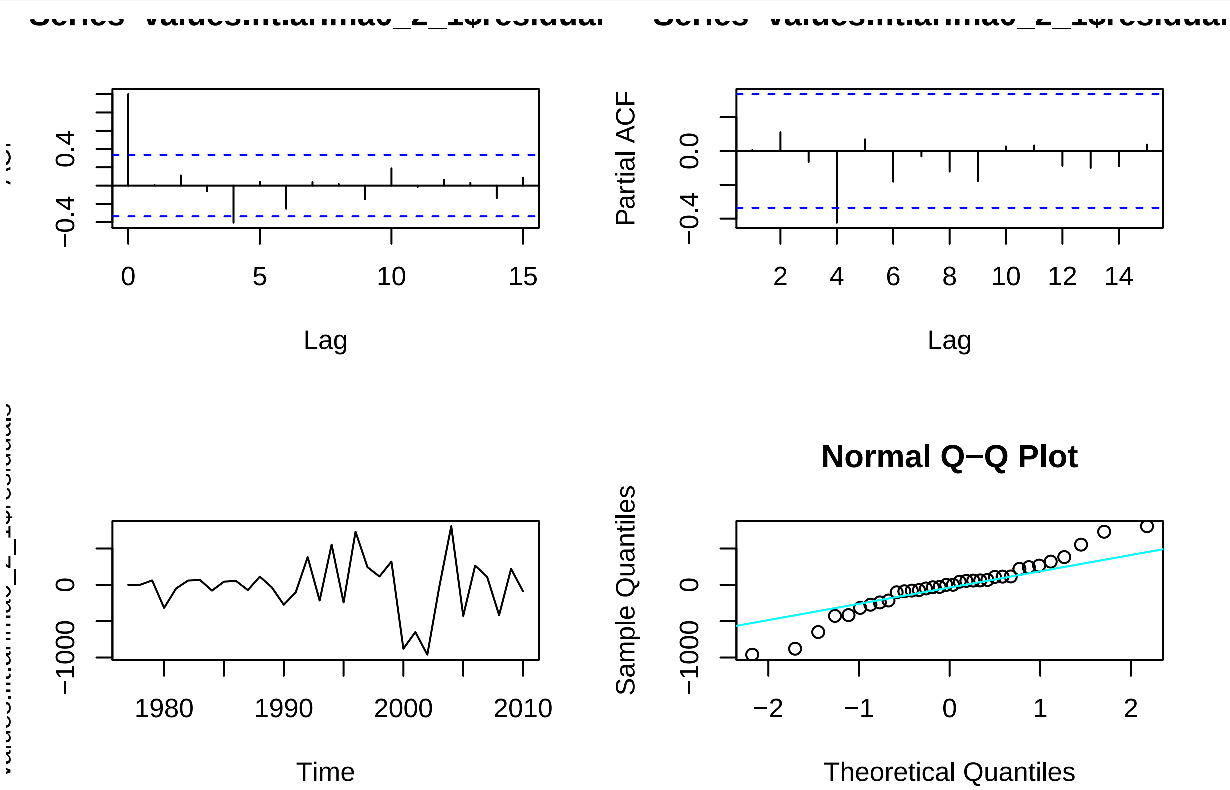
```
##
## Call:
## arima(x = values, order = c(0, 2, 1))
##
## Coefficients:
##          ma1
##       0.0525
## s.e.  0.1572
##
## sigma^2 estimated as 148807:  log likelihood = -235.97,  aic = 475.95
##
## Training set error measures:
##                     ME    RMSE      MAE        MPE     MAPE      MASE
```

```
## Training set -20.31403 374.238 264.2031 -0.3118855 4.687148 0.5772316
##                      ACF1
## Training set 0.005052652
```

```r
accuracy(values.fit.arima0_2_1)
```

```
##                     ME    RMSE      MAE        MPE     MAPE      MASE
## Training set -20.31403 374.238 264.2031 -0.3118855 4.687148 0.5772316
##                      ACF1
## Training set 0.005052652
```

```r
#arima 021 residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima0_2_1$residuals)
pacf(values.fit.arima0_2_1$residuals)
plot(values.fit.arima0_2_1$residuals)
qqnorm(values.fit.arima0_2_1$residuals)
qqline(values.fit.arima0_2_1$residuals, col="cyan")
```




```r
values.fit.arima0_2_2 <- arima(values, order=c(0,2,2)) #holt
values.fit.arima0_2_4 <- arima(values, order=c(0,2,4))


summary(values.fit.arima0_2_2)
```

```
##
## Call:
## arima(x = values, order = c(0, 2, 2))
##
## Coefficients:
##           ma1     ma2
```

```
##       0.0774  0.2994
## s.e.  0.1580  0.2107
##
## sigma^2 estimated as 141790:  log likelihood = -235.3,  aic = 476.59
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE     MAPE     MASE
## Training set -14.07232 365.3068 259.4646 -0.1882817 4.782089 0.566879
##                      ACF1
## Training set -0.008007343
```

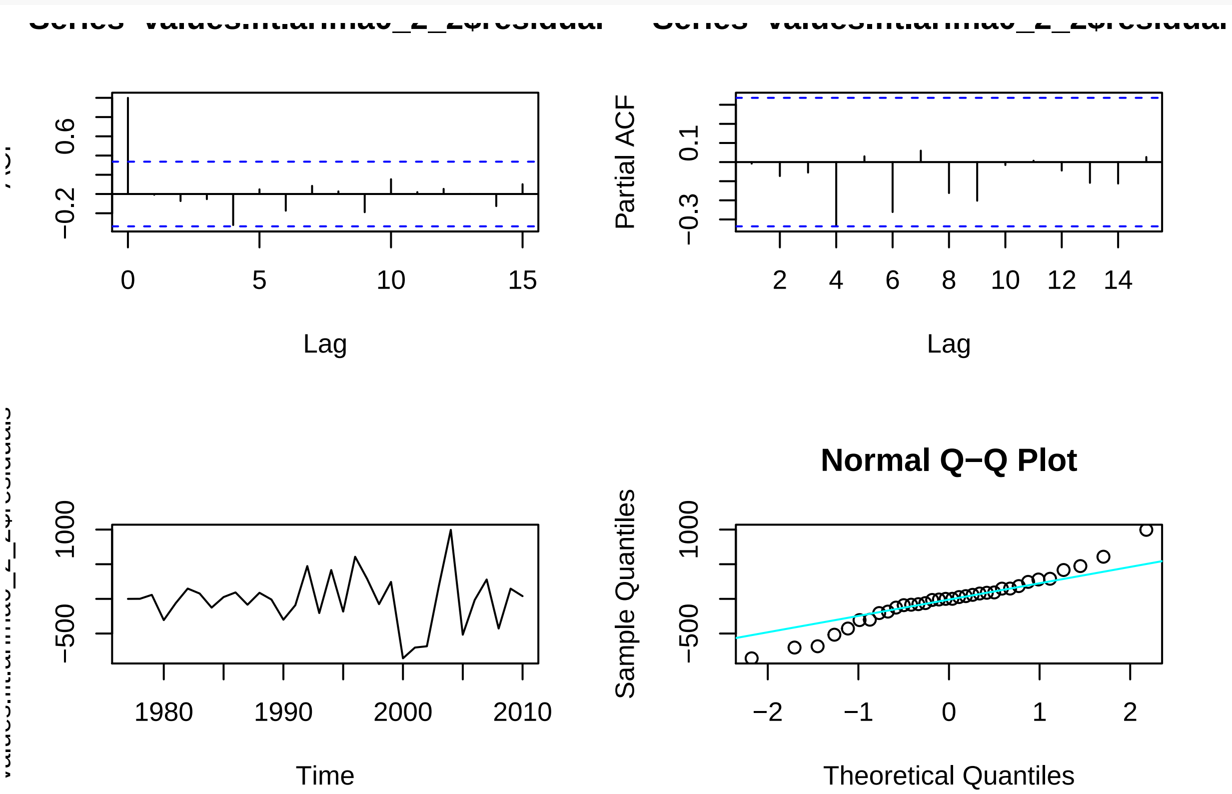
```
accuracy(values.fit.arima0_2_2)
```

```
##                     ME     RMSE      MAE        MPE     MAPE     MASE
## Training set -14.07232 365.3068 259.4646 -0.1882817 4.782089 0.566879
##                      ACF1
## Training set -0.008007343
```

```
# BIC
AIC(values.fit.arima0_2_2, k=log(length(values)))
```

```
## [1] 481.1697
```

```
#arima 021 residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima0_2_2$residuals)
pacf(values.fit.arima0_2_2$residuals)
plot(values.fit.arima0_2_2$residuals)
qqnorm(values.fit.arima0_2_2$residuals)
qqline(values.fit.arima0_2_2$residuals, col="cyan")
```



```r
summary(values.fit.arima0_2_4)
```

```
##
## Call:
## arima(x = values, order = c(0, 2, 4))
##
## Coefficients:
##           ma1      ma2      ma3      ma4
##       -0.1132  -0.1680  -0.1955  -0.5233
## s.e.   0.3837   0.3359   0.3096   0.2348
##
## sigma^2 estimated as 111282:  log likelihood = -232.74,  aic = 475.47
##
## Training set error measures:
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -30.02746 323.6294 236.8156 -0.599699 4.333539 0.5173952
##                    ACF1
## Training set 0.05459344
```

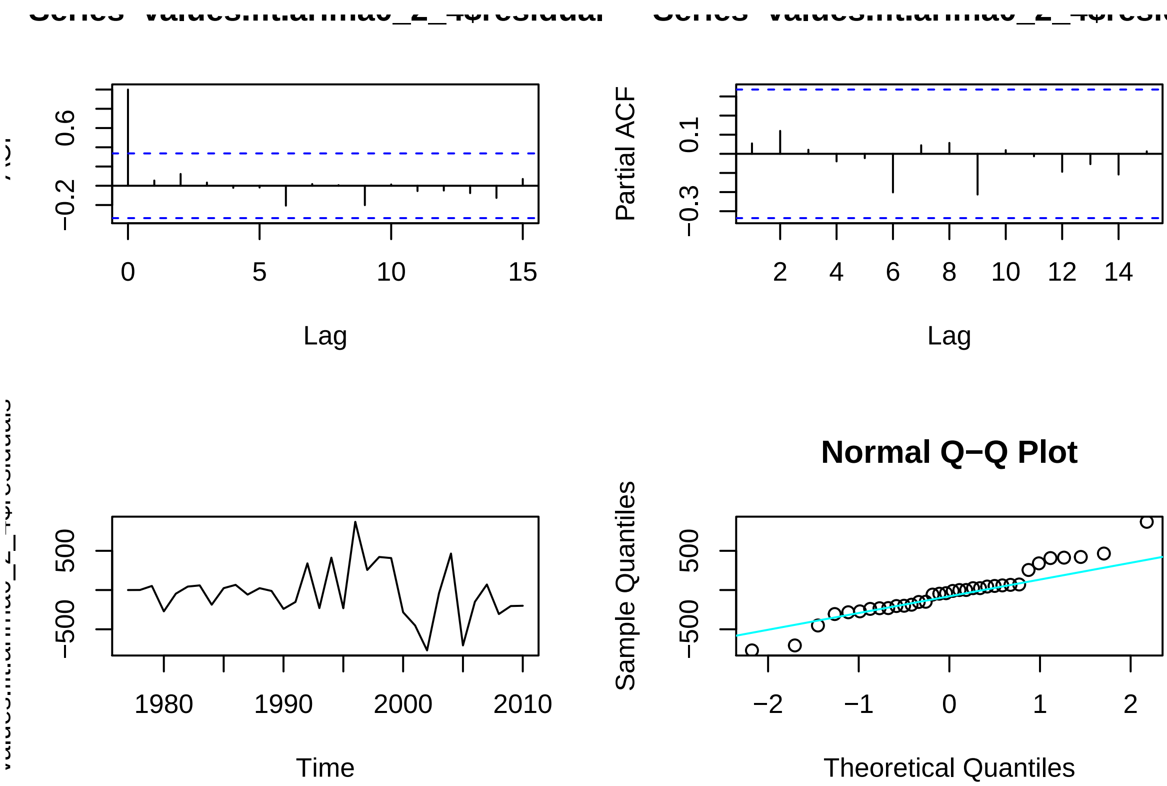
```r
accuracy(values.fit.arima0_2_4)
```

```
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -30.02746 323.6294 236.8156 -0.599699 4.333539 0.5173952
##                    ACF1
## Training set 0.05459344
```

```r
# BIC
AIC(values.fit.arima0_2_4, k=log(length(values)))
```

```
## [1] 483.1039
```

```r
#arima residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima0_2_4$residuals)
pacf(values.fit.arima0_2_4$residuals)
plot(values.fit.arima0_2_4$residuals)
qqnorm(values.fit.arima0_2_4$residuals)
qqline(values.fit.arima0_2_4$residuals, col="cyan")
```



maybe something in between…

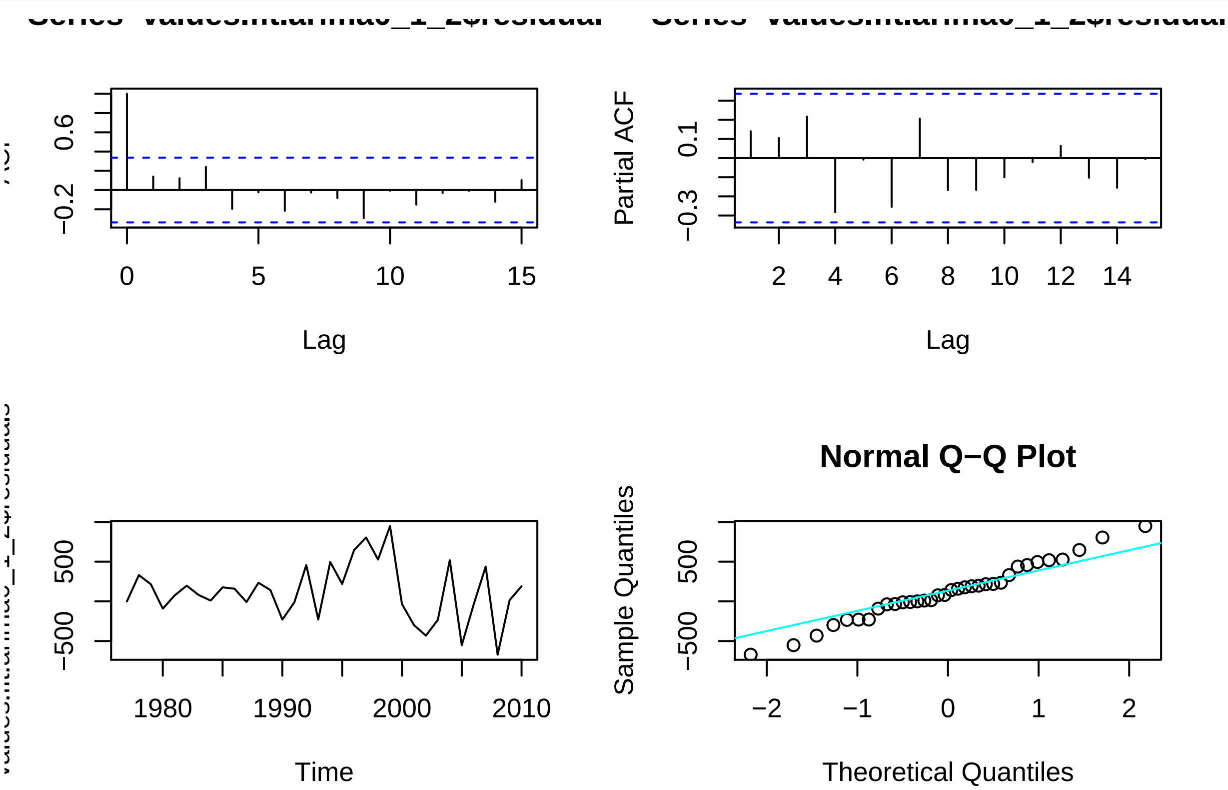
```r
values.fit.arima0_1_2 <- arima(values, order=c(0,1,2))


summary(values.fit.arima0_1_2)
```

```
##
## Call:
## arima(x = values, order = c(0, 1, 2))
##
## Coefficients:
##          ma1     ma2
##       0.7317  0.6806
## s.e.  0.1524  0.1397
##
## sigma^2 estimated as 144844:  log likelihood = -243.63,  aic = 493.26
##
## Training set error measures:
##                     ME    RMSE      MAE      MPE     MAPE      MASE
## Training set 119.2386 374.945 285.6024 4.517042 6.890703 0.6239848
##                    ACF1
## Training set 0.1405702
```

```r
accuracy(values.fit.arima0_1_2)
```

```
##                     ME    RMSE      MAE      MPE     MAPE      MASE
## Training set 119.2386 374.945 285.6024 4.517042 6.890703 0.6239848
##                    ACF1
## Training set 0.1405702
```

```r
#arima 012 residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima0_1_2$residuals)
pacf(values.fit.arima0_1_2$residuals)
plot(values.fit.arima0_1_2$residuals)
qqnorm(values.fit.arima0_1_2$residuals)
qqline(values.fit.arima0_1_2$residuals, col="cyan")
```



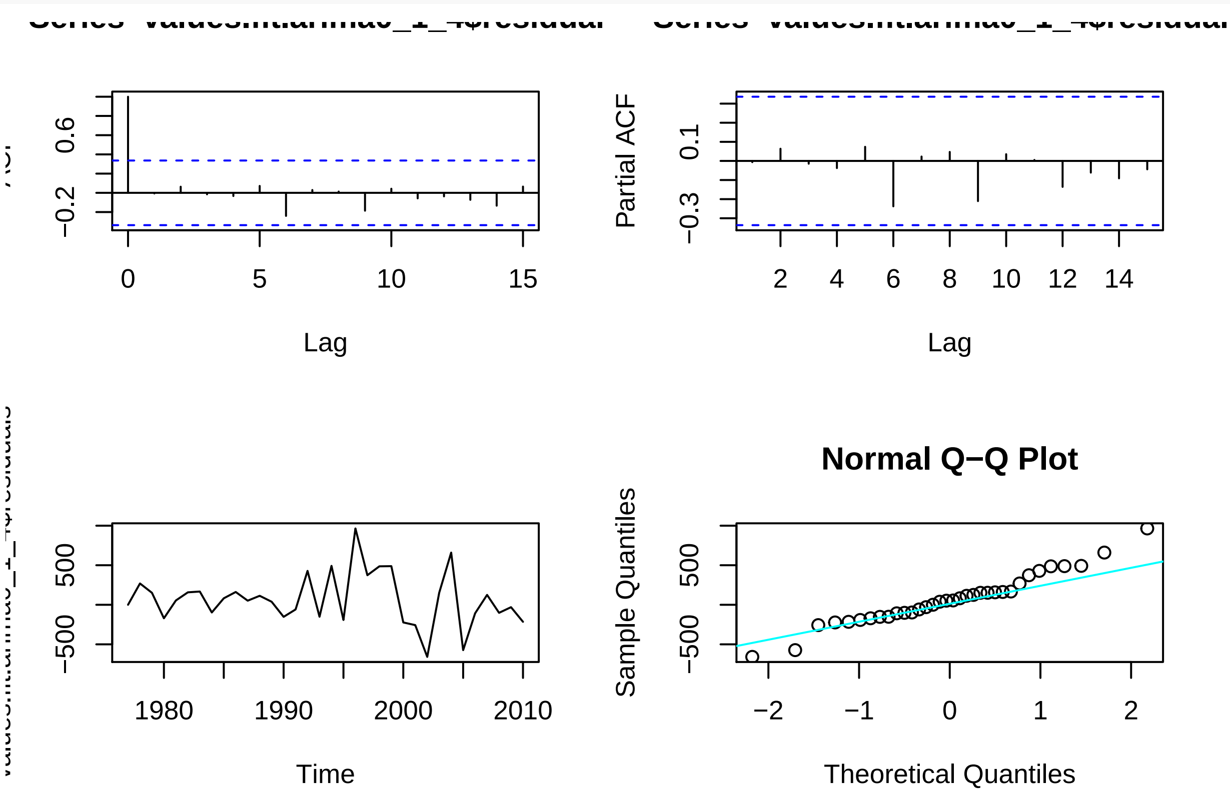

```r
values.fit.arima0_1_4 <- arima(values, order=c(0,1,4))


summary(values.fit.arima0_1_4)
```

```
##
## Call:
## arima(x = values, order = c(0, 1, 4))
##
## Coefficients:
##          ma1     ma2     ma3     ma4
##       0.9239  0.8149  0.6900  0.1416
## s.e.  0.1663  0.1821  0.2226  0.1652
##
## sigma^2 estimated as 112714:  log likelihood = -239.65,  aic = 489.3
##
## Training set error measures:
##                    ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 71.34951 330.7549 247.3252 3.126773 6.089426 0.5403567
##                      ACF1
## Training set -0.006407687
```

```r
accuracy(values.fit.arima0_1_4)
```

```
##                    ME     RMSE      MAE      MPE     MAPE      MASE
## Training set 71.34951 330.7549 247.3252 3.126773 6.089426 0.5403567
##                      ACF1
## Training set -0.006407687
```

```r
#arima 021 residuals analysis
par(mfrow=c(2,2))
acf(values.fit.arima0_1_4$residuals)
pacf(values.fit.arima0_1_4$residuals)
plot(values.fit.arima0_1_4$residuals)
qqnorm(values.fit.arima0_1_4$residuals)
qqline(values.fit.arima0_1_4$residuals, col="cyan")
```



Model Comparison

Although we are able to compute AIC for both ARIMA and ETS models, we can't directly use it to compare between ETS and ARIMA models because they are in different model classes, and the likelihood is computed in different ways. Instead, we can use time series cross validation:

```r
fets <- function(x, h) {
  forecast(ets(x), h = h)
}
farima <- function(x, h) {
  forecast(auto.arima(x), h=h)
}

# Compute CV errors for ETS
res.ets <- tsCV(values, fets, h=5)
```

```r
# Compute CV errors for ARIMA
res.autoarima <- tsCV(values, fautoarima, h=5)

# Find MSE of each model class
mean(res.ets^2, na.rm=TRUE)
```

```
## [1] 4335414
```

```r
mean(res.autoarima^2, na.rm=TRUE)
```

```
## [1] NaN
```

ETS produced MSE = 4,335,414 While ARIMA's MSE = 4,780,749

ETS performs better by this metric

For final selected models

```r
fets <- function(x, h) {
  forecast(ets(x, model="MAN"), h = h)
}
farima <- function(x, h, order) {
  forecast(arima(x, order = c(0,2,2)), h = h)
}

# Compute CV errors for ETS
res.MANets <- tsCV(values, fets, h=1)
# Compute CV errors for ARIMA
res.arima0_2_2 <- tsCV(values, farima, h=1)

# Find MSE of each model class
sqrt(mean(res.MANets^2, na.rm=TRUE))
```

```
## [1] 633.9152
```

```r
sqrt(mean(res.arima0_2_2^2, na.rm=TRUE))
```

```
## [1] 429.8334
```

```r
res.MANets
```

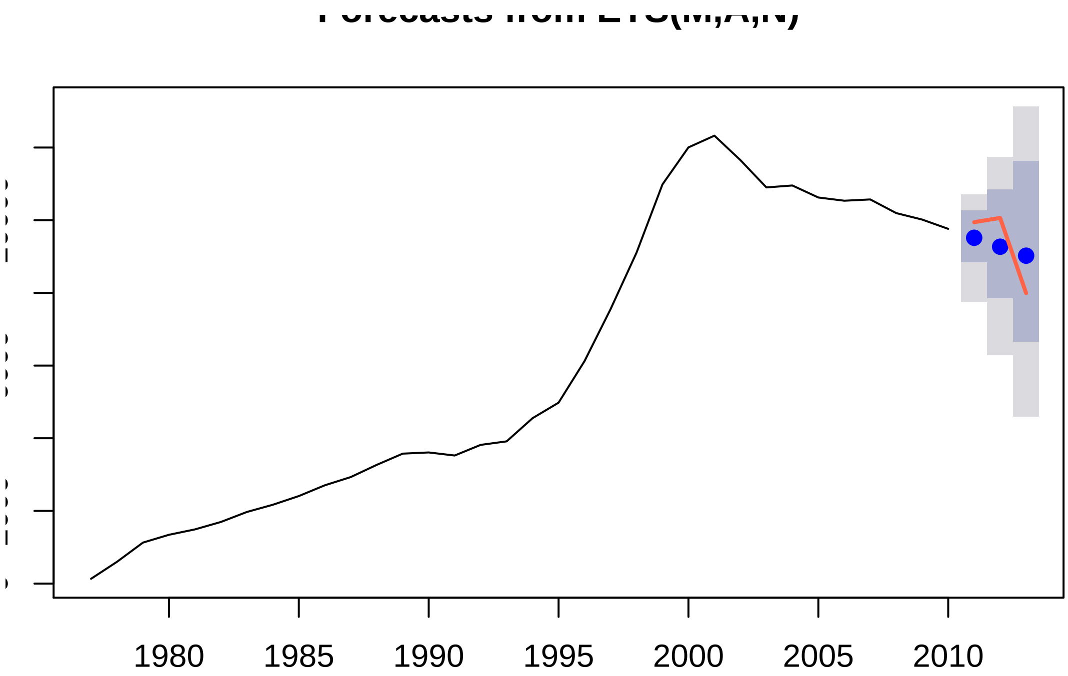
```
## Time Series:
## Start = 1977
## End = 2010
## Frequency = 1
##  [1]    467.78000   232.20110  -253.06000  -247.06892  -167.52034
##  [6]    -47.91137   -26.21266    36.30136   -10.95227   -48.57561
## [11]     42.40322    51.93237  -195.15364  -512.19833  -405.85107
## [16] -1023.62280   576.58927   148.50160  1025.99423  2129.01628
## [21]    358.16889   351.66185  -795.44979  -664.09745 -1026.93520
## [26] -1276.57283  -454.49837  -711.15904   202.42262   197.87506
## [31]  -391.51300  -628.65941  -141.83663          NA
```

Forecasting

ETS

```r
forecast.fit.MANets <- forecast(values.fit.MANets, h=3, bootstrap = TRUE)
plot(forecast.fit.MANets)
```

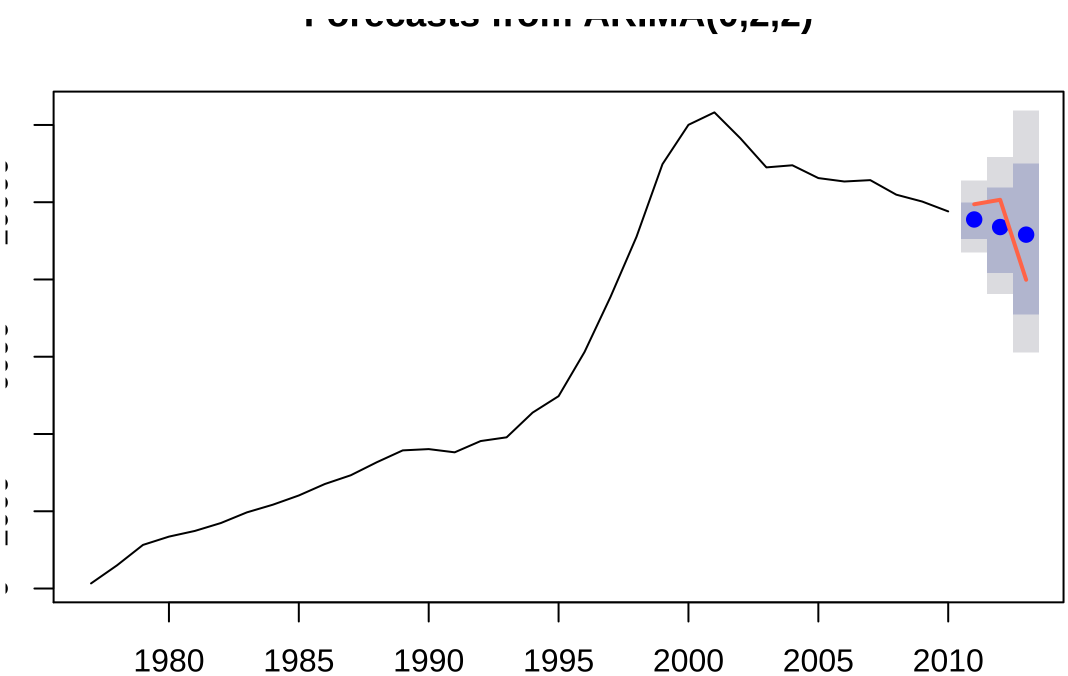
```r
lines(values.holdout, col="tomato", lwd=2)
```




```r
accuracy(forecast.fit.MANets, values.holdout)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -23.19990 381.9091 265.6755 -0.2811852 4.762199 0.5804484
## Test set      63.26275 791.2311 751.5192 -0.2407782 8.371060 1.6419210
##                    ACF1 Theil's U
## Training set  0.2136499        NA
## Test set     -0.2845866 0.6301251
```

```r
forecast.fit.arima0_2_2 <- forecast(values.fit.arima0_2_2, h=3, bootstrap = TRUE)
plot(forecast.fit.arima0_2_2)
lines(values.holdout, col="tomato", lwd=2)
```




```r
accuracy(forecast.fit.arima0_2_2, values.holdout)
```

```
##                      ME      RMSE      MAE        MPE     MAPE     MASE
## Training set -14.07232 365.3068 259.4646 -0.1882817 4.782089 0.566879
## Test set     -23.35957 820.2937 755.8974 -1.2201806 8.530307 1.651487
##                      ACF1 Theil's U
## Training set -0.008007343        NA
## Test set     -0.263299122 0.6603451
```

prediction intervals:

```r
values.holdout
```

```
## Time Series:
## Start = 2011
## End = 2013
## Frequency = 1
##         Value
## [1,]  9946.93
## [2,] 10062.58
## [3,]  7991.98
```

```r
forecast.fit.MANets
```

```
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2011       9516.770 8843.701 10263.06 7743.145 10711.72
## 2012       9270.567 7852.716 10837.79 6297.899 11742.74
## 2013       9024.365 6660.404 11628.70 4600.614 13136.36
```

```r
forecast.fit.arima0_2_2
```

```
##      Point Forecast    Lo 80     Hi 80    Lo 95    Hi 95
## 2011       9553.514 9050.203  9983.831 8709.390 10563.77
## 2012       9357.190 8164.181 10371.798 7634.530 11169.23
## 2013       9160.866 7090.152 10993.362 6113.315 12373.86
```