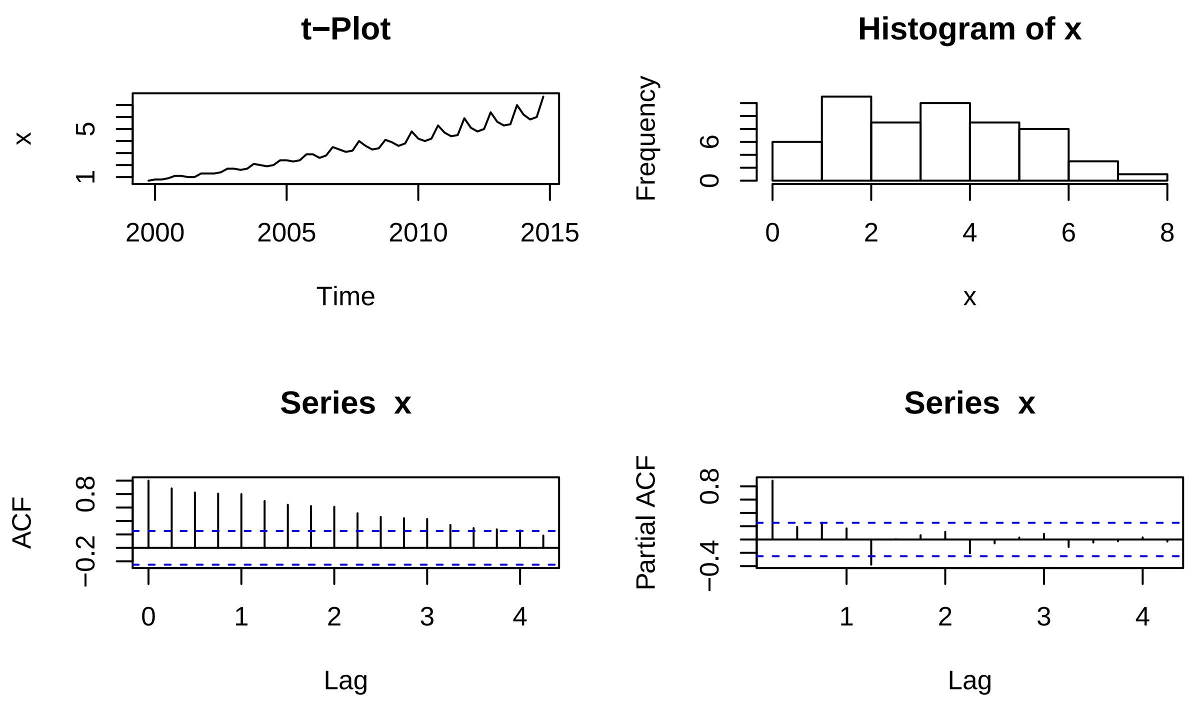
Statistical Methods for Discrete Response, Time Series, and Panel Data (W271): Lab 3

Professor Jeffrey Yau

Question 1: Forecasting using a SARIMA model

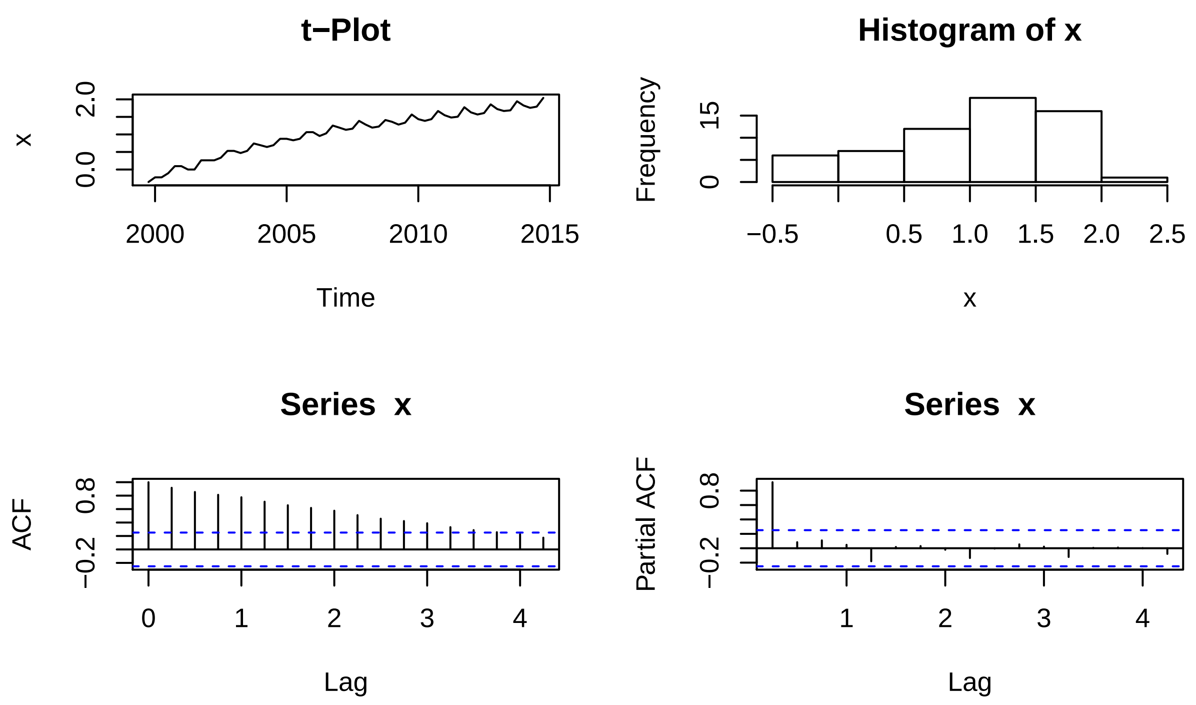
Note: Custom function ts_plots() [plot time series], ts_resid() [plot residuals], forecast_exp() [exponentiate all values in forecasts objects] and arimatable() [summarizes arima table] are not included in the R pdf but is in the R-markdown file.

```r
df=read.csv('ECOMPCTNSA.csv')
head(df)
# exclude 2015 & 2016
dfts=ts(df$ECOMPCTNSA,start=c(1999,4),end=c(2014,4),freq=4)
ts_plots(dfts)
```



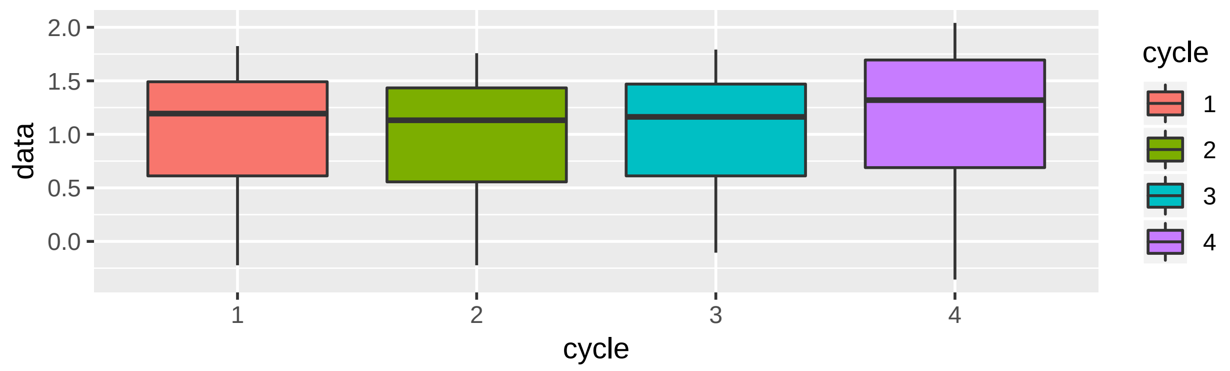
There is clearly a seaonsonal component based on the plot. Additionally, the plot seems to be potentially heteroskedastic. We will log the time series to decrease the heteroskedasticity and re-analyze the time series from the starting point.

```r
dfts_log=log(dfts)
ts_plots(dfts_log)
```



From the logged time series plots, there is a trend and seasonal component. The seasonality is likely to be quarterly from the t-plot. We will look at a box plot of the data by season.

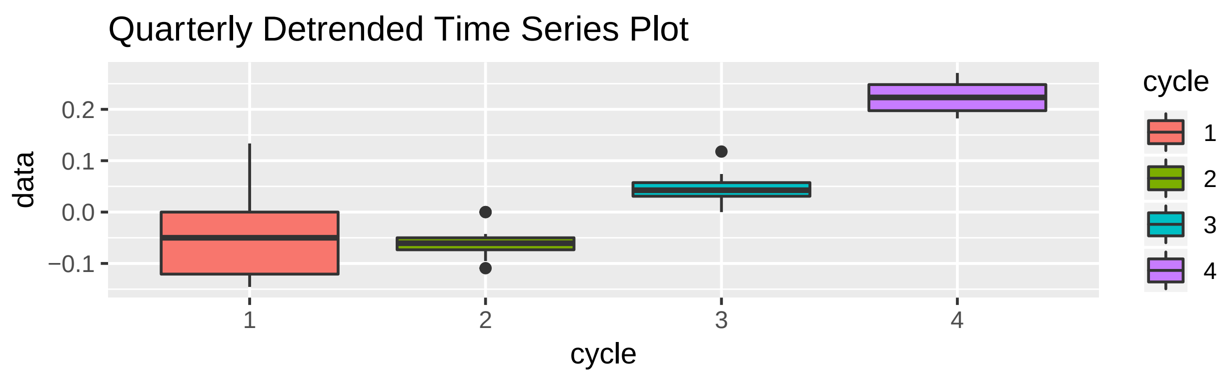
```
ggplot(data.frame(cycle=factor(cycle(dfts_log)),data=as.numeric(dfts_log)),aes(x=cycle,y=data,g
```



The seasonal boxplot does not show significant differences in each of the quarters. However, this is possibly due to the trend component affecting the cycles. We will detrend the series first and re-exmine the quarterly plots.

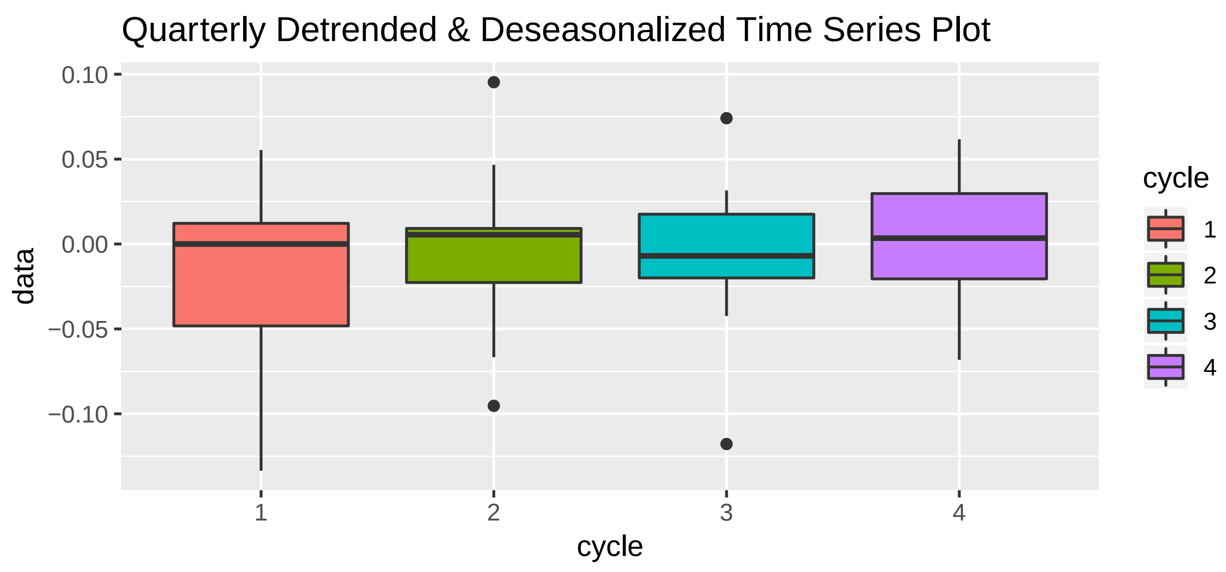
First, the time series will be detrended using a first differencing and we examine the seasonality plot.

```
tmp=diff(dfts_log,lag=1)
ggplot(data.frame(cycle=factor(cycle(tmp)),data=as.numeric(tmp)),aes(x=cycle,y=data,group=cycle
```



The detrended series shows a strong quarterly seasonality. The time series will be deseasonalized using a quarterly cycle.

```
df_ds=diff(diff(dfts_log,lag=1),lag=4)
ggplot(data.frame(cycle=factor(cycle(df_ds)),data=as.numeric(df_ds)),aes(x=cycle,y=data,group=
```



```
adf.test(df_ds)
```

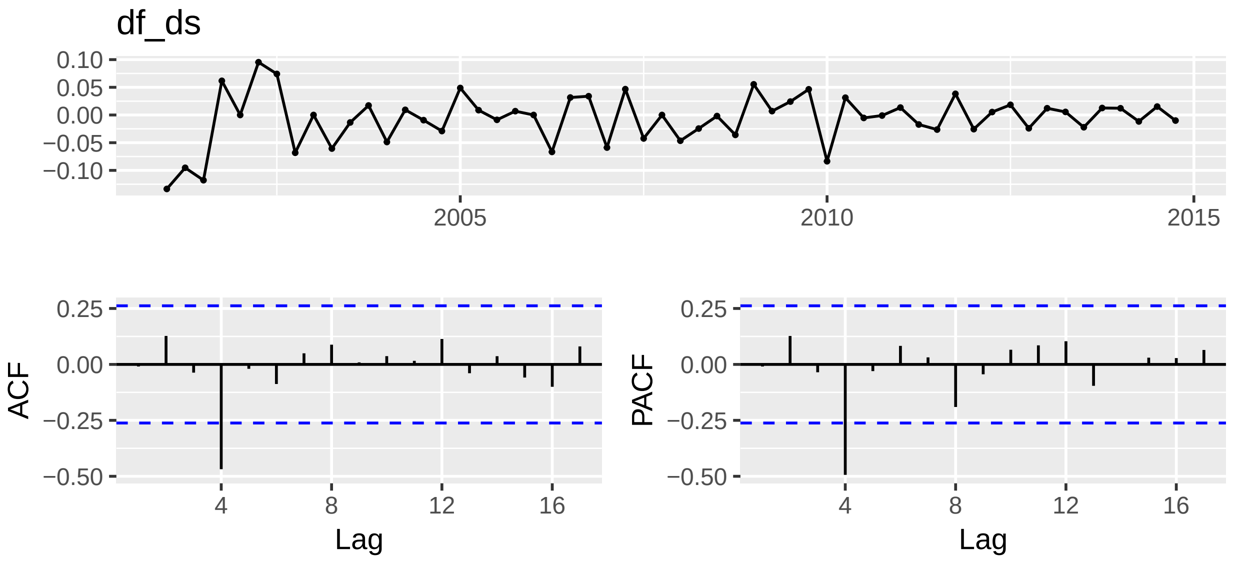
```
##
##  Augmented Dickey-Fuller Test
##
## data:  df_ds
## Dickey-Fuller = -7.2107, Lag order = 3, p-value = 0.01
## alternative hypothesis: stationary
```

```
pp.test(df_ds)
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  df_ds
## Dickey-Fuller Z(alpha) = -52.65, Truncation lag parameter = 3,
## p-value = 0.01
## alternative hypothesis: stationary
```

The detrended and deseasonalized plot show the quarterly mean and variance is now similar across the quarters indicating a deseasonalized time series. Augmented Dickey-Fuller and Phillips-Perron test are performed on deseasonalized, detrended time series both rejecting the non-stationary hypothesis. With a stationary time series, we can use an ARMA model to model the detrended, deseasonalized time series. Since the there are $I(1)$ and $I(1)_4$ components in the time series, we will use the SARIMA model to model the original logged time series rather than modeling the detrended and deseasonalized logged data to combine the steps.

```
ggtsdisplay(df_ds)
```



–SS Do we use the deseasonalized, detrended data to decide where to start with our SARIMA? or the original data?

The t-plot shows the time series to have been detrended and deseasonalized. On the pacf plot, there appears to strong serial correlation quarterly as it oscillates towards zero. The acf plot has high serial correlation at the 4th lag. The strong acf at lag 4 and cycling towards 0 in pact suggest there is a SMA(1) component.

Based on our exploration, the intial model will be $SARIMA(0,1,0)(0,1,1)_4$.

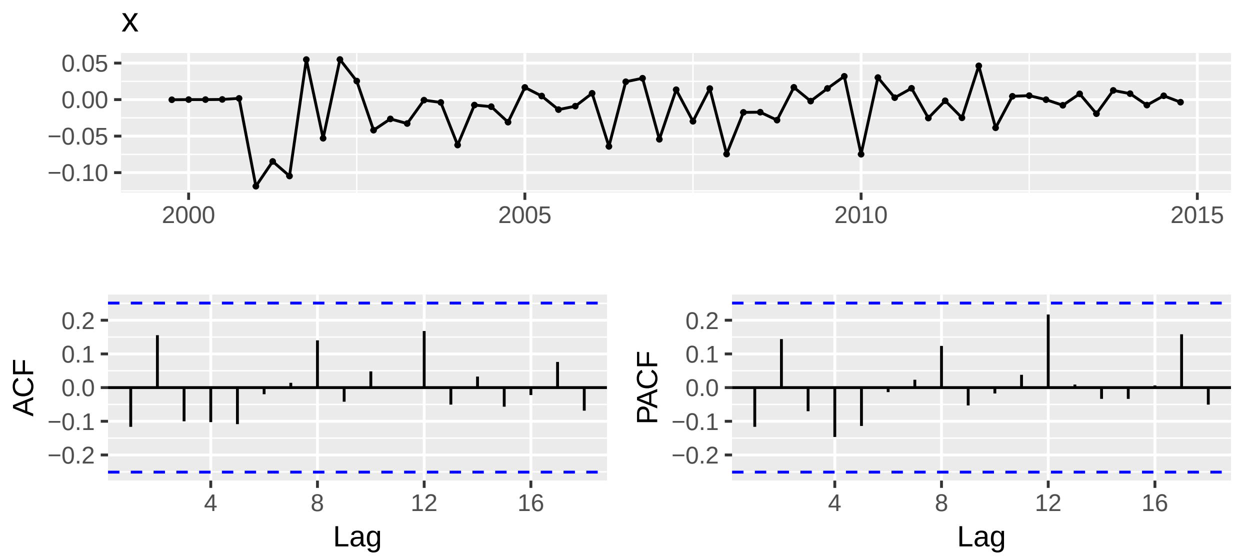
```
m = arima(dfts_log,order=c(0,1,0),seasonal = list(order=c(0,1,1),period=4))
xtable(arimatable(m,dfts_log))
```

	beta	SE	Sigma2	AIC	BIC	LogLik	ME	RMSE	MAE
sma1	-0.52	0.10	0.00	-201.54	-197.49	102.77	-0.01	0.04	0.03

The SMA1 β at -0.5167 with se. at 0.0975 suggesting signficance. The β_{sma1} does not cross zero.

The t-plot appears to be white noise with heteroskedasticity. We will not further attempt to fit the variance of the residuals with GARCH/ARCH models. The t-plot shows no trend or seasonality. The residuals appear to be a stationary white noise process (though not necessarily Gaussian white noise) from the augmented Dickey Fuller and Phillips-Perron Test. Finally, the Ljung-box test almost shows the residuals to be uncorrelated with a p-Value of 0.35 as confirmed by the visual inspection.

```r
ts_resid(m$residuals)
```



	PhillipsPerron	AugmentedDickeyFuller	LjungBox
p-Value	0.01	0.01	0.35

We will examine other $SARIMA(p,1,q)(P,1,Q)_4$ up to $p = q = P = Q = 2$ to aid in choosing our final model.

```r
results=data.frame(matrix(ncol=9,nrow=0))
for (p in 0:2){ for (q in 0:2) { for (PS in 0:2) { for (QS in 0:2) {
        m.tmp=arima(dfts_log,order=c(p,1,q),seasonal = list(order=c(PS,1,QS),period=4))
        results=rbind(results,c(p=p,d=1, q=q,P=PS,D=1,Q=QS,AIC=AIC(m.tmp),BIC=BIC(m.tmp),Log_Li
colnames(results) = c('p','d','q','P','D','Q','AIC','BIC','Log_Likelihood')
xtable(head(results[order(results$AIC,results$BIC),],5))
```

	p	d	q	P	D	Q	AIC	BIC	Log_Likelihood
3	0.00	1.00	0.00	0.00	1.00	2.00	-207.22	-201.14	106.61
7	0.00	1.00	0.00	2.00	1.00	0.00	-206.50	-200.42	106.25
4	0.00	1.00	0.00	1.00	1.00	0.00	-206.22	-202.17	105.11
30	1.00	1.00	0.00	0.00	1.00	2.00	-205.53	-197.42	106.76
5	0.00	1.00	0.00	1.00	1.00	1.00	-205.50	-199.42	105.75

beta SE Sigma2 AIC BIC LogLik sar1 -0.8011532 0.1506089 0.001302963 -206.4998 -200.4238 106.2499 sar2 -0.2490731 0.1618362 0.001302963 -206.4998 -200.4238 106.2499 ME RMSE MAE sar1 -0.01010292 0.03396238 0.02369047 sar2 -0.01010292 0.03396238 0.02369047

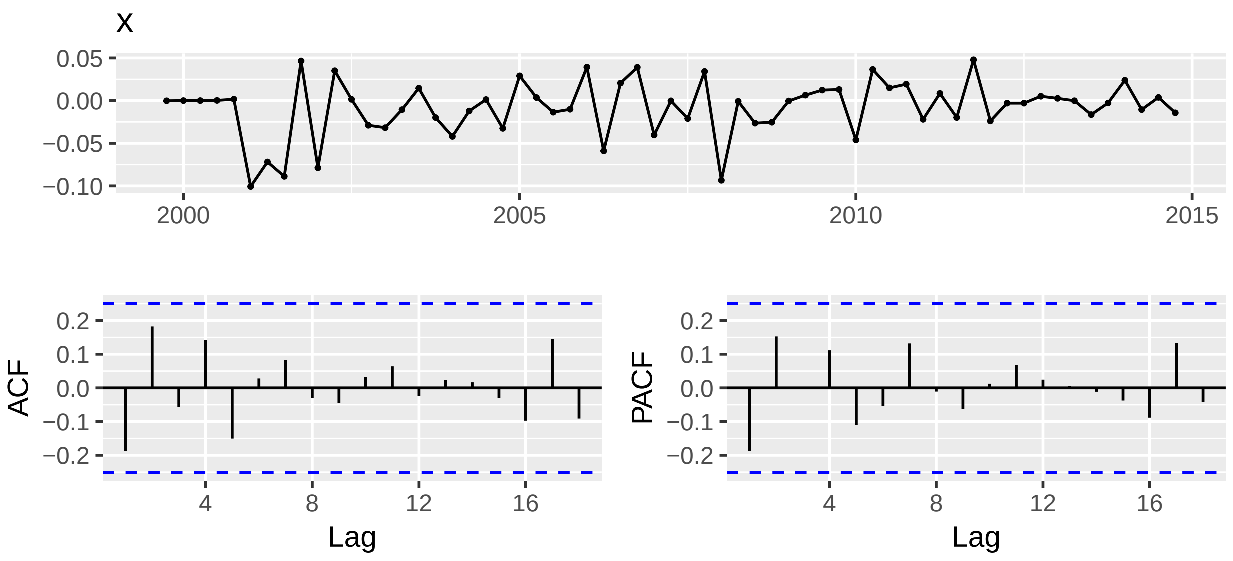
Auto.arima() selected $SARIMA(0,1,0)(2,1,0)_4$. From the manual iterations and auto.arims(), $SARIMA(0,1,0)(0,1,2)_4$ and $SARIMA(0,1,0)(2,1,0)_4$ are chosen as the candidate models as they have the lowest AICs and BICs.

In the $SARIMA(0,1,0)(0,1,2)_4$, βs appear to be statistically siginficant and the residuals do appear to be white noise with stationarity and no autocorrelation. The Ljung-Box p-Value is higher than the $SARIMA(0,1,0)(0,1,1)_4$ model. Note that the heteroskedasticity in residuals no longer appear.

```r
m.010012=arima(dfts_log,order=c(0,1,0),seasonal = list(order=c(0,1,2),period=4))
xtable(arimatable(m.010012,dfts_log))
```

	beta	SE	Sigma2	AIC	BIC	LogLik	ME	RMSE	MAE
sma1	-0.77	0.16	0.00	-207.22	-201.14	106.61	-0.01	0.03	0.02
sma2	0.40	0.13	0.00	-207.22	-201.14	106.61	-0.01	0.03	0.02

```r
ts_resid(m.010012$residuals)
```



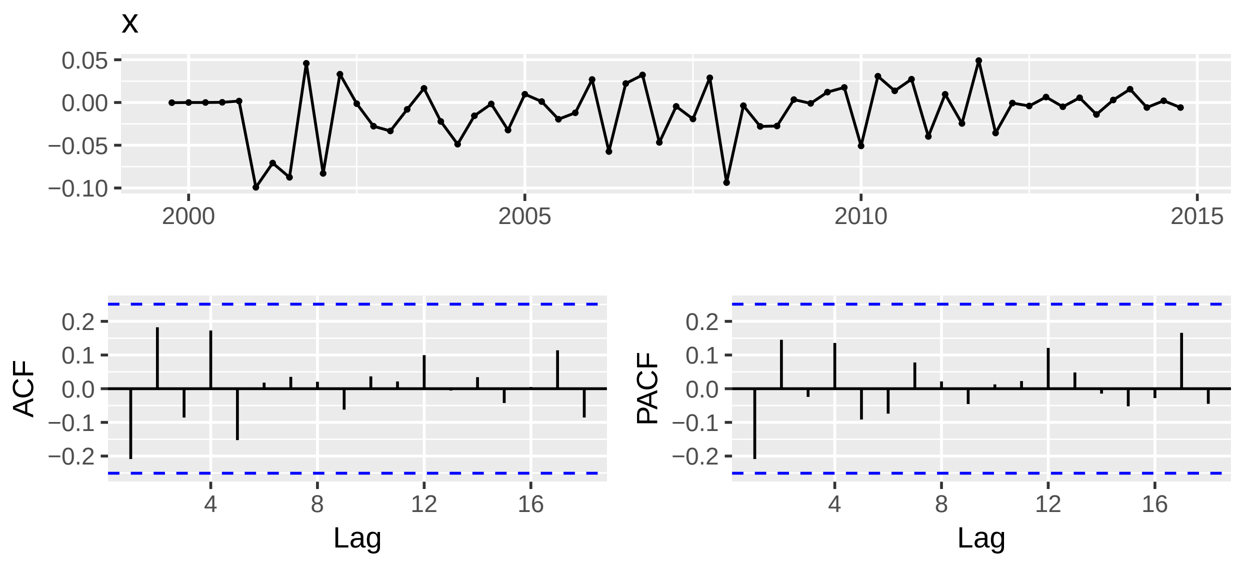
	PhillipsPerron	AugmentedDickeyFuller	LjungBox
p-Value	0.01	0.01	0.13

In the $SARIMA(0,1,0)(2,1,0)_4$, the βs do not include zero up to the 95% confidence interval and the residual appear to be stationary and white noise. The Ljung-Box p-Value is ???? and the heterskedasticity of the residuals is not evident.

```r
m.010210=arima(dfts_log,order=c(0,1,0),seasonal = list(order=c(2,1,0),period=4))
xtable(arimatable(m.010210,dfts_log))
```

	beta	SE	Sigma2	AIC	BIC	LogLik	ME	RMSE	MAE
sar1	-0.80	0.15	0.00	-206.50	-200.42	106.25	-0.01	0.03	0.02
sar2	-0.25	0.16	0.00	-206.50	-200.42	106.25	-0.01	0.03	0.02

```
ts_resid(m.010210$residuals)
```

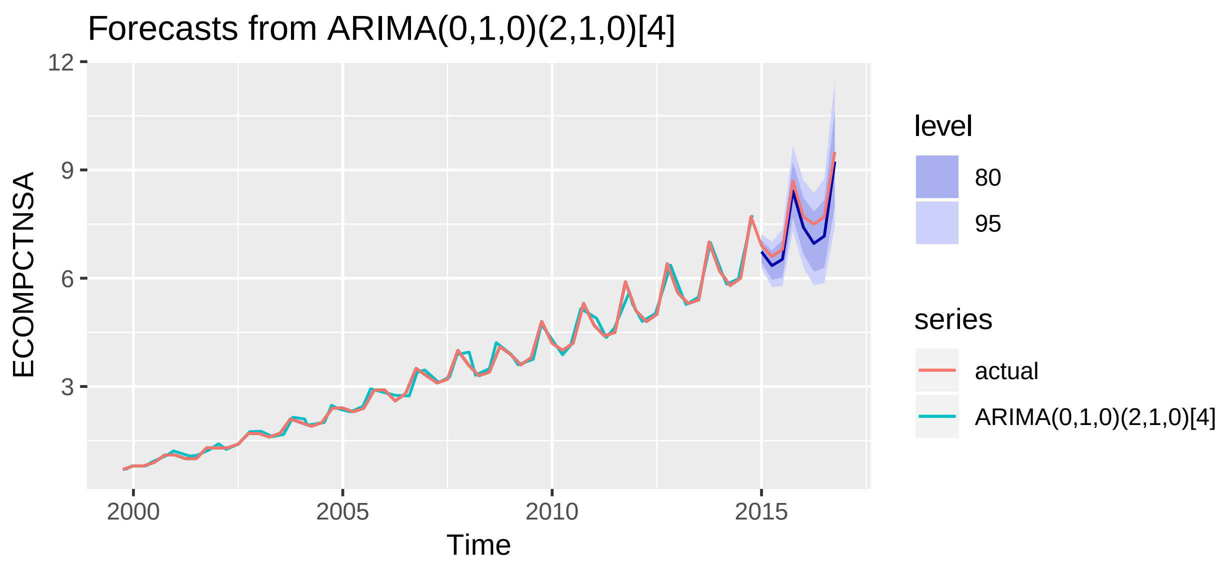


	PhillipsPerron	AugmentedDickeyFuller	LjungBox
p-Value	0.01	0.01	0.09

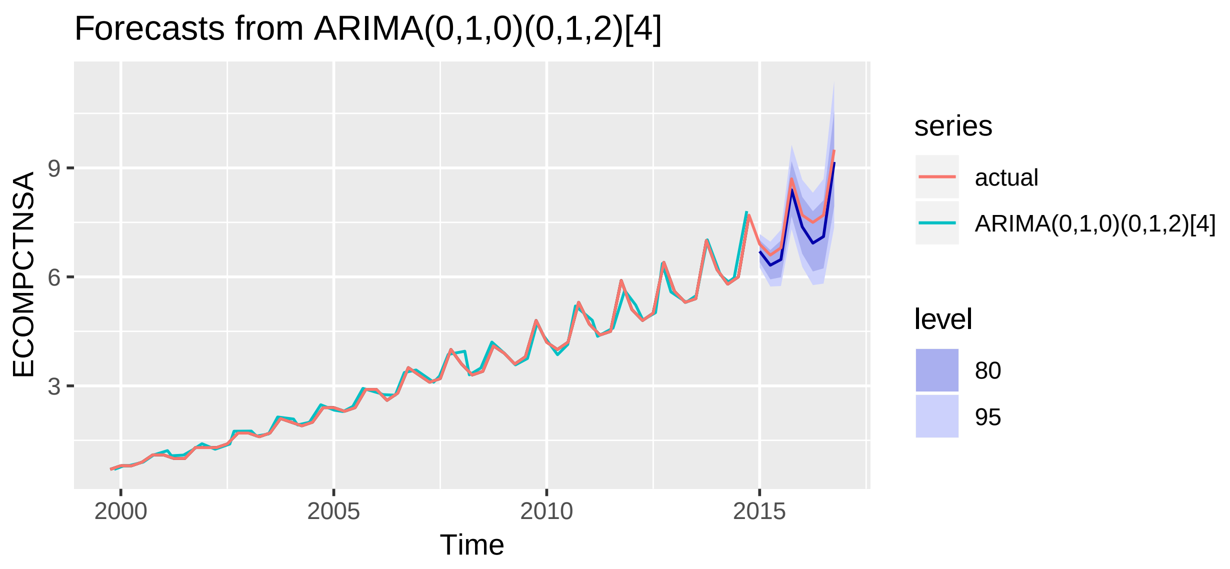
We first note that these series seem comparable as a MA model can be inverted to become an AR model. An $SARIMA(0,1,0)(0,1,2)_4$ is very similar to $SARIMA(0,1,0)(2,1,0)_4$ from invertibility of MA models.

The 2 models will be chosen as potential candidates. We will examine both in-sample and out-out-sample fits to chose the final model.

```
actual=ts(df$ECOMPCTNSA,start=c(1999,4),freq=4)
forecast_sar=forecast_exp_func(forecast(m.010210))
forecast_sma=forecast_exp_func(forecast(m.010012))
autoplot(forecast_sar) +autolayer(exp(fitted(m.010210)),series='ARIMA(0,1,0)(2,1,0)[4]',positi
```



```
autoplot(forecast_sma) +autolayer(exp(fitted(m.010012)),series='ARIMA(0,1,0)(0,1,2)[4]',positi
```



The in-sample fits for both models are extremely close to the historical fit. The predictions for both models are extremely similar. We will select the models based on accuracy of the time series. The time series is logged to avoid overweighting the larger values on the time series due to the trend.

```
pred_test=window(log(actual),start=c(2015,1))
xtable(accuracy(forecast(m.010210),pred_test),caption='$ARIMA(0,1,0)(2,1,0)_4$')
```

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Theil's U
Training set	-0.01	0.03	0.02	-Inf	Inf	0.16	-0.21	
Test set	0.04	0.05	0.04	2.17	2.17	0.30	0.18	0.37

Table 1: $ARIMA(0,1,0)(2,1,0)_4$

```
xtable(accuracy(forecast(m.010012),pred_test),caption='$ARIMA(0,1,0)(0,1,2)_4$')
```

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Theil's U
Training set	-0.01	0.03	0.02	-Inf	Inf	0.16	-0.19	
Test set	0.05	0.05	0.05	2.44	2.44	0.34	0.20	0.41

Table 2: $ARIMA(0,1,0)(0,1,2)_4$

Every accuracy measure tested showed a lower error with $SARIMA(0,1,0)(2,1,0)_4$ model. The final model chosen is # CHECK BELOW

$$SARIMA(0,1,0)(2,1,0)_4$$

$$(1-0.7851B-0.23651B^2)_4(1-B)_4(1-B)x_t=\epsilon_t$$

$$y_t=y_{t-1}+y_{t-4}+-0.7851y_{t-4}+-0.23651y_{t-5}+\epsilon_t$$

```
df_full_log=ts(log(df$ECOMPCTNSA),start=c(1999,4),freq=4)
m=arima(df_full_log,order=c(0,1,0),seasonal = list(order=c(2,1,0),period=4))
xtable(arimatable(m,df_full_log))
```

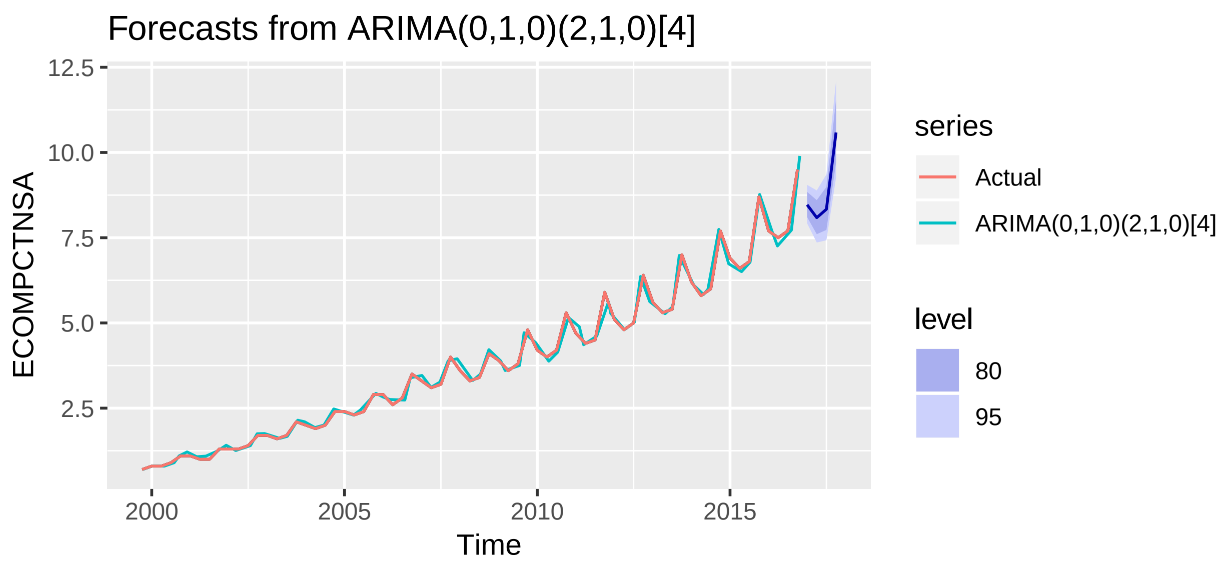
	beta	SE	Sigma2	AIC	BIC	LogLik	ME	RMSE	MAE
sar1	-0.79	0.14	0.00	-242.53	-236.05	124.27	-0.01	0.03	0.02
sar2	-0.24	0.16	0.00	-242.53	-236.05	124.27	-0.01	0.03	0.02

The forecast for 2017 using $SARIMA(0,1,0)(2,1,0)_4$ is

```
forecast_sar=forecast_exp_func(forecast(m,h = 4))
xtable(data.frame(forecast_sar))
```

	Point.Forecast	Lo.80	Hi.80	Lo.95	Hi.95
2017 Q1	8.47	8.11	8.84	7.92	9.05
2017 Q2	8.09	7.60	8.60	7.36	8.89
2017 Q3	8.33	7.73	8.99	7.43	9.36
2017 Q4	10.59	9.70	11.55	9.27	12.10

```
autoplot(forecast_sar,'Model') +autolayer(exp(fitted(m)),series='ARIMA(0,1,0)(2,1,0)[4]',positi
```

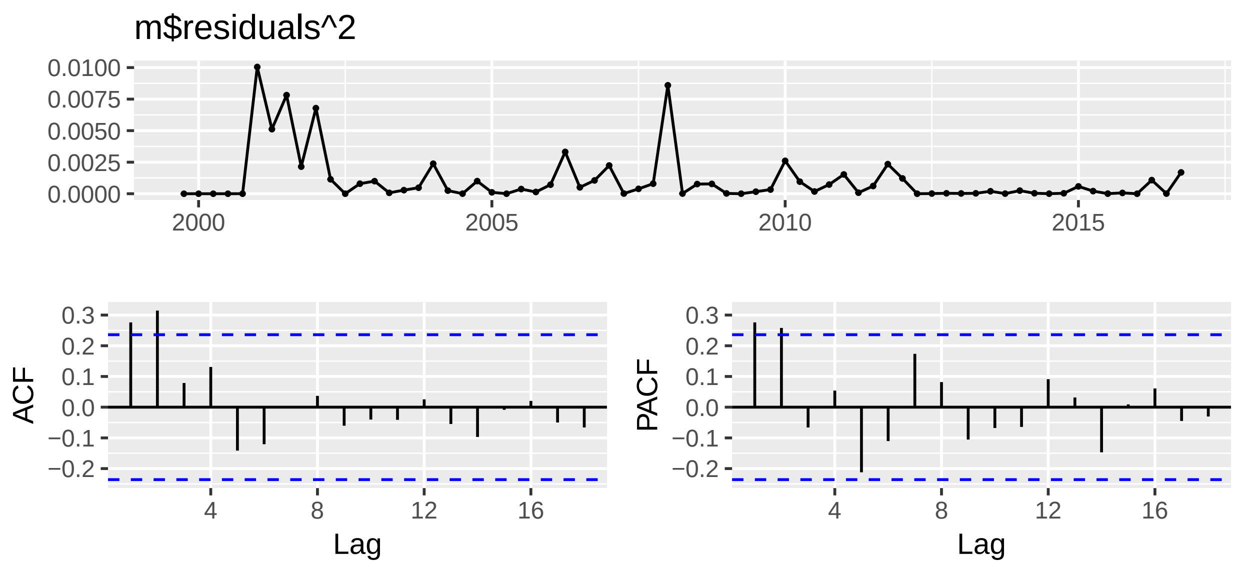


KYLE - OWN TEST

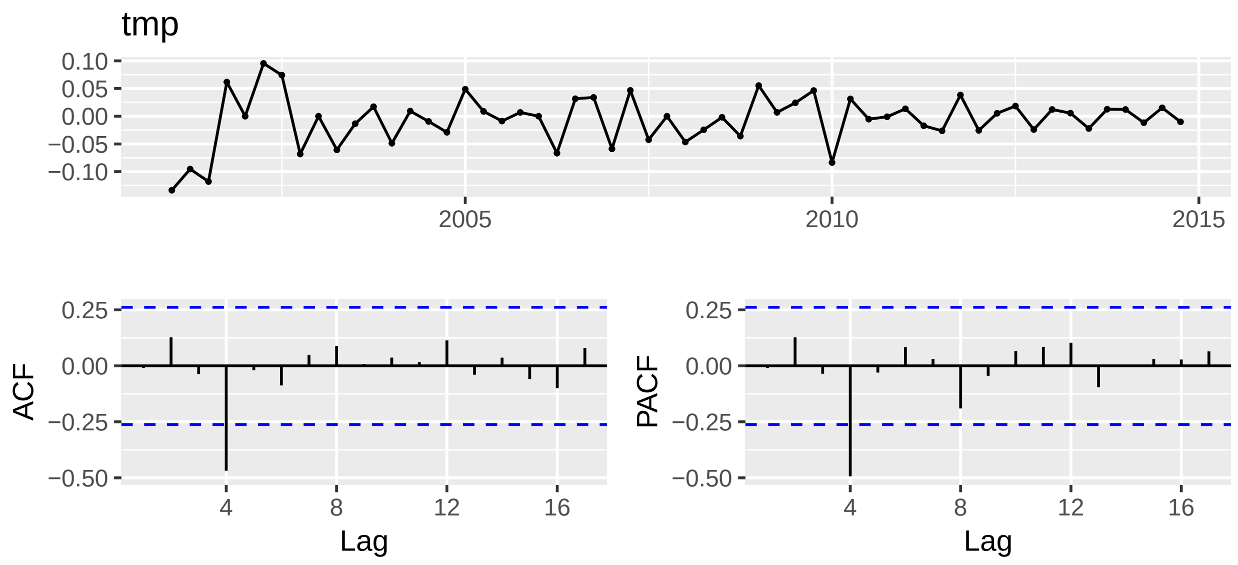
Furthermore, the residuals appear to be heteroskedastic. From the $residuals^2$ act and pacf plots, There appears to be an AR component in the volatility suggesting a GARCH(1,0) model.

```
#library(fGarch)
#m.garch=garchFit(~garch(1,0),m$residuals,trace=FALSE)
#summary(m.garch)
```

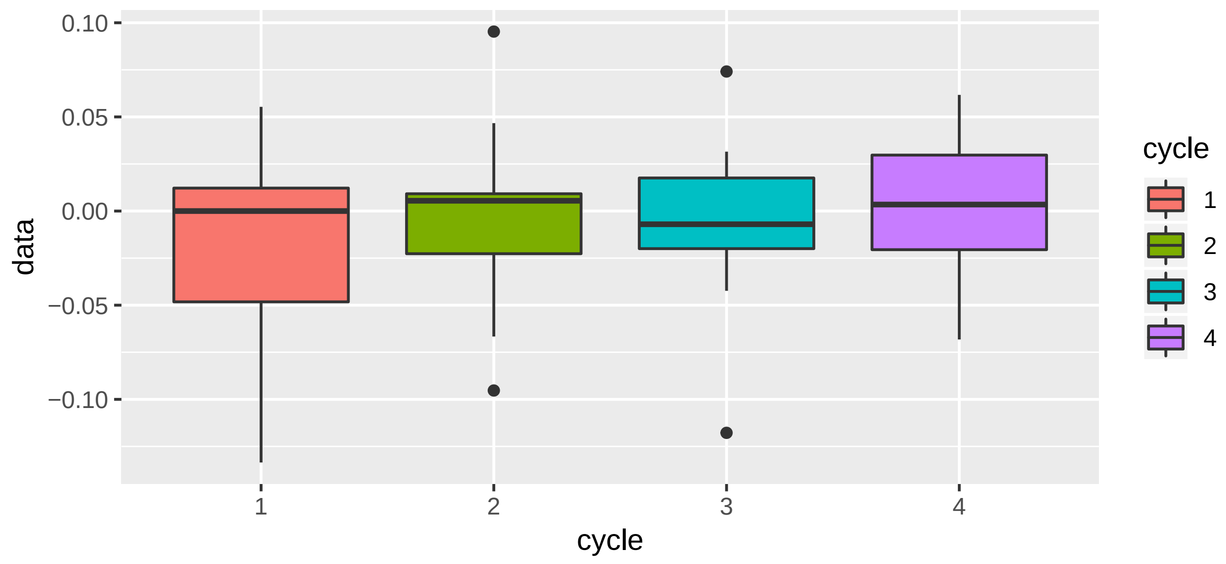
```
ggtsdisplay(m$residuals^2)
```



```
tmp=diff(diff(dfts_log,lag=1),lag=4)
ggtsdisplay(tmp)
```



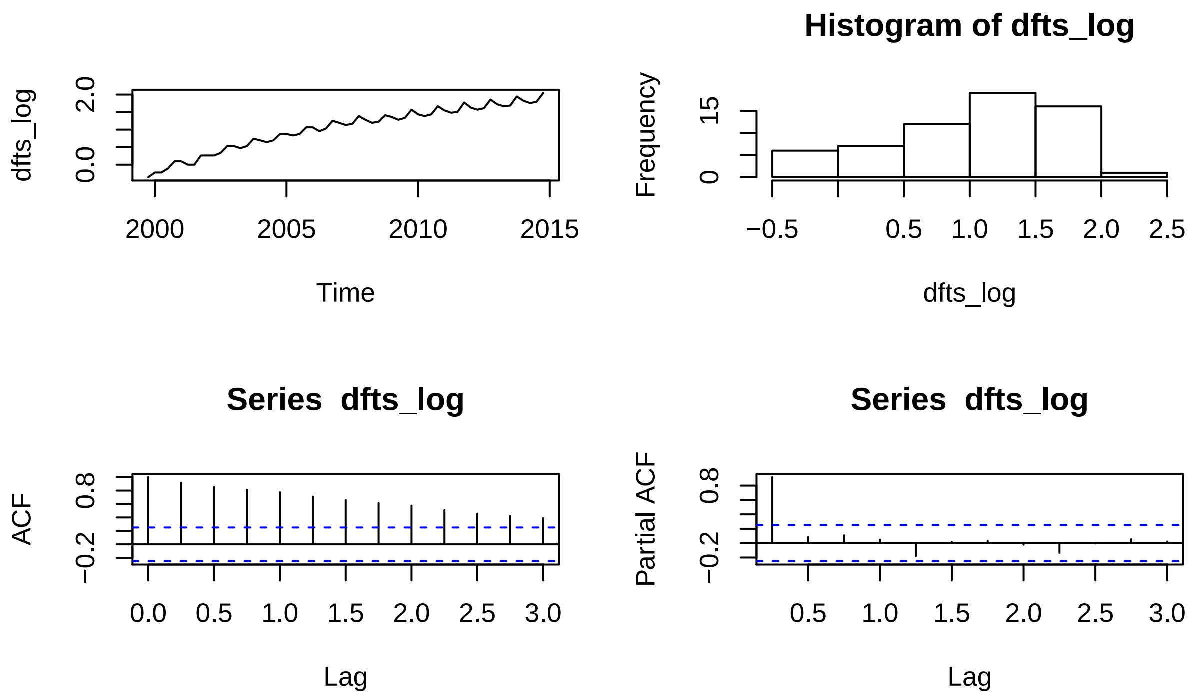
```
ggplot(data.frame(cycle=factor(cycle(tmp)),data=as.numeric(tmp)),aes(x=cycle,y=data,group=cycl
```



```
#ggplot(data.frame(r=residuals(m.garch)),aes(sample=r))+geom_qq()+geom_qq_line(col='red')
#hist(residuals(m.garch))
#predict(m.garch)
```

The series appear to be detrended and deseaonalized. From the ACF gaph

```
par(mfrow=c(2,2))
plot(dfts_log)
hist(dfts_log)
acf(dfts_log,lag=12)
pacf(dfts_log,lag=12)
```



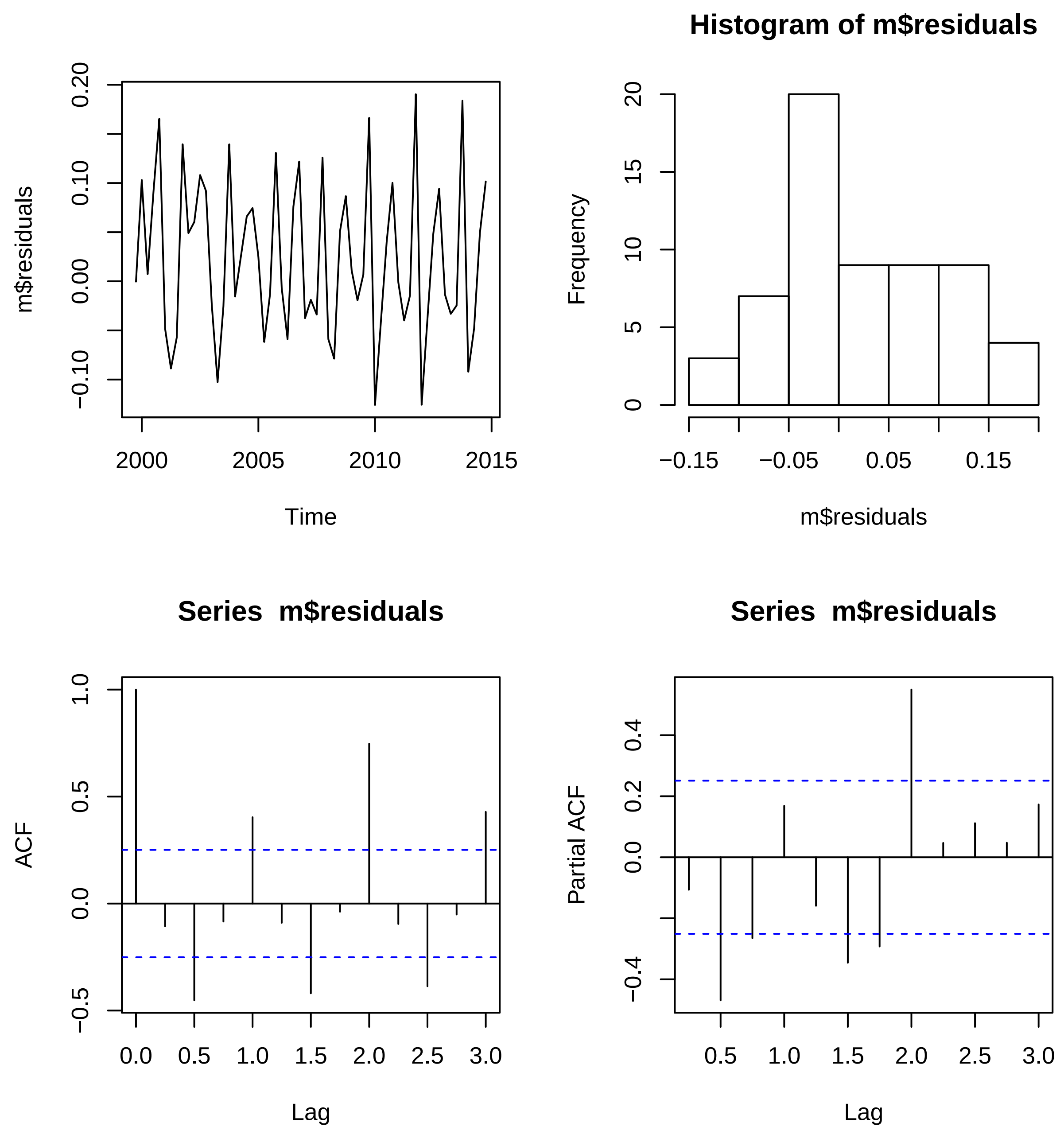
The initial model proposed is a $SARIMA(1,1,0)(0,0,1)_4$ model.

```
m=arima(dfts_log,order=c(1,1,0),seasonal=list(order=c(0,0,1),period=4))
```

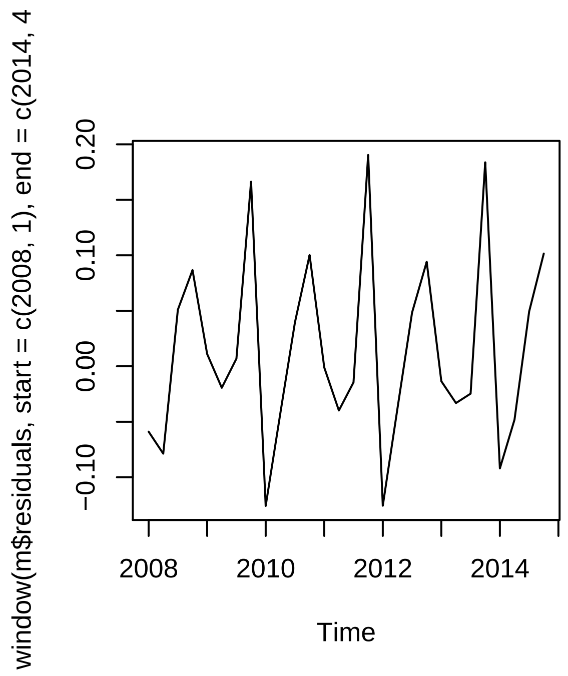
The AR(1) component is likely to be 0 given the -0.0705 β and s.e. of 0.1347. The SMA(1) β appears siginificant. We will examine the residuals to see if the I(1) has removed the trend.

From the t-plot, the I(1) appears to have removed the trend. The seasonality does not seem to be removed but m

```
par(mfrow=c(2,2))
plot(m$residuals)
hist(m$residuals)
acf(m$residuals,lag=12)
pacf(m$residuals,lag=12)
```



```
plot(window(m$residuals,start=c(2008,1),end=c(2014,4)))
```


window(m$residuals, start = c(2008, 1), end = c(2014, 4
0.20
0.10
0.00
−0.10
2008
2010
2012
2014
Time

```r
library(ggplot2)
```

Question 2: Learning how to use the xts library

Materials covered in Question 2 of this lab

- Primarily the references listed in this document:
  - "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich. 2008. (xts.pdf)
  - "xts FAQ" by xts Development Team. 2013 (xts_faq.pdf)
  - xts_cheatsheet.pdf

Task 1:

1. Read A. The **Introduction** section (Section 1), which only has 1 page of reading of xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich B. The first three questions in"xts FAQ" a. What is xts? b. Why should I use xts rather than zoo or another time-series package? c. HowdoIinstallxts? C. The "A quick introduction to xts and zoo objects" section in this document

2. Read the "A quick introduction to xts and zoo objects" of this document

A quick introduction to xts and zoo objects

xts

`xts` - stands for eXtensible Time Series - is an extended zoo object - is essentially matrix + (time-based) index (aka, observation + time)

- xts is a constructor or a subclass that inherits behavior from parent (zoo); in fact, it extends the popular zoo class. As such. most zoo methods work for xts
- is a matrix objects; subsets always preserve the matrix form
- importantly, xts are indexed by a formal time object. Therefore, the data is time-stamped
- The two most important arguments are `x` for the data and `order.by` for the index. `x` must be a vector or matrix. `order.by` is a vector of the same length or number of rows of `x`; it must be a proper time or date object and be in an increasing order

Task 2:

1. Read A. Section 3.1 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich

   B. The following questions in "xts FAQ" a. How do I create an xts index with millisecond precision? b. OK, so now I have my millisecond series but I still canâ t see the milliseconds displayed. What went wrong?

2. Follow the following section of this document

Creating an xts object and converting to an xts object from an imported dataset

We will create an **xts** object from a matrix and a time index. First, let's create a matrix and a time index. The matrix, as it creates, is not associated with the time indext yet.

```r
# Set working directory
#wd <-"~/Documents/Teach/Cal/w271/_2018.03_Fall/labs/lab3"
#setwd(wd)
```

```r
# Create a matrix
x <- matrix(rnorm(200), ncol=2, nrow=100)
colnames(x) <- c("Series01", "Series02")
str(x)
```

num [1:100, 1:2] 1.42 0.524 -2.345 1.828 0.614 … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:2] "Series01" "Series02"

```r
head(x,10)
```

```
    Series01   Series02
```

[1,] 1.4195405 1.2860139 [2,] 0.5243293 -0.4196913 [3,] -2.3448217 -1.2379349 [4,] 1.8279942 -0.1532421 [5,] 0.6135923 0.3558965 [6,] 1.3723711 -0.1365458 [7,] 0.1111175 -1.3389139 [8,] 0.4449314 -0.8420480 [9,] 1.4564247 -0.9712233 [10,] -0.3216833 0.2594527

```r
idx <- seq(as.Date("2015/1/1"), by = "day", length.out = 100)
str(idx)
```

Date[1:100], format: "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05" …

```r
head(idx)
```

[1] "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05" [6] "2015-01-06"

```r
tail(idx)
```

[1] "2015-04-05" "2015-04-06" "2015-04-07" "2015-04-08" "2015-04-09" [6] "2015-04-10"

In a nutshell, **xts** is a matrix indexed by a time object. To create an xts object, we "bind" the object with the index. Since we have already created a matrix and a time index (of the same length as the number of rows of the matrix), we are ready to "bind" them together. We will name it *X*.

```r
library(xts)
X <- xts(x, order.by=idx)
str(X)
```

An 'xts' object on 2015-01-01/2015-04-10 containing: Data: num [1:100, 1:2] 1.42 0.524 -2.345 1.828 0.614 … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:2] "Series01" "Series02" Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```r
head(X,10)
```

```
           Series01   Series02
```

2015-01-01 1.4195405 1.2860139 2015-01-02 0.5243293 -0.4196913 2015-01-03 -2.3448217 -1.2379349 2015-01-04 1.8279942 -0.1532421 2015-01-05 0.6135923 0.3558965 2015-01-06 1.3723711 -0.1365458 2015-01-07 0.1111175 -1.3389139 2015-01-08 0.4449314 -0.8420480 2015-01-09 1.4564247 -0.9712233 2015-01-10 -0.3216833 0.2594527 As you can see from the structure of an `xts` objevct, it contains both a data component and an index, indexed by an objevct of class `Date`.

xtx constructor

```
xts(x=Null,
    order.by=index(x),
    frequency=NULL,
    unique=NULL,
    tzone=Sys.getenv("TZ"))
```

As mentioned previous, the two most important arguments are `x` and `order.by`. In fact, we only use these two arguments to create a xts object before.

With a xts object, one can decompose it.

Deconstructing xts

`coredata()` is used to extract the data component

```
head(coredata(X),5)
```

```
   Series01   Series02
```

[1,] 1.4195405 1.2860139 [2,] 0.5243293 -0.4196913 [3,] -2.3448217 -1.2379349 [4,] 1.8279942 -0.1532421 [5,] 0.6135923 0.3558965

`index()` is used to extract the index (aka times)

```
head(index(X),5)
```

[1] "2015-01-01" "2015-01-02" "2015-01-03" "2015-01-04" "2015-01-05"

Conversion to xts from other time-series objects

We will use the same dataset "bls_unemployment.csv" that we used in the last live session to illustarte the functions below.

```
df <- read.csv("bls_unemployment.csv", header=TRUE, stringsAsFactors = FALSE)

# Examine the data structure
  str(df)
```

'data.frame': 121 obs. of 4 variables: $ Series.id: chr "LNU04000000" "LNU04000000" "LNU04000000" "LNU04000000" ... $ Year : int 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ... $ Period : chr "M01" "M02" "M03" "M04" ... $ Value : num 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ...

```
  names(df)
```

[1] "Series.id" "Year" "Period" "Value"

```
  head(df)
```

```
Series.id Year Period Value
```

1 LNU04000000 2007 M01 5.0 2 LNU04000000 2007 M02 4.9 3 LNU04000000 2007 M03 4.5 4 LNU04000000 2007 M04 4.3 5 LNU04000000 2007 M05 4.3 6 LNU04000000 2007 M06 4.7

```
  tail(df)
```

```
  Series.id Year Period Value
```

116 LNU04000000 2016 M08 5.0 117 LNU04000000 2016 M09 4.8 118 LNU04000000 2016 M10 4.7 119 LNU04000000 2016 M11 4.4 120 LNU04000000 2016 M12 4.5 121 LNU04000000 2017 M01 5.1

```
#table(df$Series.id, useNA = "always")
#table(df$Period, useNA = "always")

# Convert a column of the data frame into a time-series object
unemp <- ts(df$Value, start = c(2007,1), end = c(2017,1), frequency = 12)
  str(unemp)
```

Time-Series [1:121] from 2007 to 2017: 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 …

```
  head(cbind(time(unemp), unemp),5)
```

```
      time(unemp) unemp
```

Jan 2007 2007.000 5.0 Feb 2007 2007.083 4.9 Mar 2007 2007.167 4.5 Apr 2007 2007.250 4.3 May 2007 2007.333 4.3

```
# Now, let's convert it to an xts object
df_matrix <- as.matrix(df)
  head(df_matrix)
```

```
 Series.id     Year   Period Value
```

[1,] "LNU04000000" "2007" "M01" " 5.0" [2,] "LNU04000000" "2007" "M02" " 4.9" [3,] "LNU04000000" "2007" "M03" " 4.5" [4,] "LNU04000000" "2007" "M04" " 4.3" [5,] "LNU04000000" "2007" "M05" " 4.3" [6,] "LNU04000000" "2007" "M06" " 4.7"

```
  str(df_matrix)
```

chr [1:121, 1:4] "LNU04000000" "LNU04000000" "LNU04000000" … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:4] "Series.id" "Year" "Period" "Value"

```
  rownames(df)
```

[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" [12] "12" "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" [23] "23" "24" "25" "26" "27" "28" "29" "30" "31" "32" "33" [34] "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" [45] "45" "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" [56] "56" "57" "58" "59" "60" "61" "62" "63" "64" "65" "66" [67] "67" "68" "69" "70" "71" "72" "73" "74" "75" "76" "77" [78] "78" "79" "80" "81" "82" "83" "84" "85" "86" "87" "88" [89] "89" "90" "91" "92" "93" "94" "95" "96" "97" "98" "99" [100] "100" "101" "102" "103" "104" "105"

“106” “107” “108” “109” “110” [111] “111” “112” “113” “114” “115” “116” “117” “118” “119” “120” “121”

```
unemp_idx <- seq(as.Date("2007/1/1"), by = "month", length.out =
length(df[,1]))
  head(unemp_idx)
```

[1] “2007-01-01” “2007-02-01” “2007-03-01” “2007-04-01” “2007-05-01” [6] “2007-06-01”

```
unemp_xts <- xts(df$Value, order.by = unemp_idx)
  str(unemp_xts)
```

An ‘xts’ object on 2007-01-01/2017-01-01 containing: Data: num [1:121, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 … Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```
  head(unemp_xts)
```

```
       [,1]
```

2007-01-01 5.0 2007-02-01 4.9 2007-03-01 4.5 2007-04-01 4.3 2007-05-01 4.3 2007-06-01 4.7

Task 3:

1. Read A. Section 3.2 of “xts: Extensible Time Series” by Jeffrey A. Ryan and Joshua M. Ulrich
2. Follow the following section of this document

Merging and modifying time series

One of the key strengths of `xts` is that it is easy to join data by column and row using a only few different functions. It makes creating time series datasets almost effortless.

The important criterion is that the xts objects must be of identical type (e.g. integer + integer), or be POSIXct dates vector, or be atomic vectors of the same type (e.g. numeric), or be a single NA. It does not work on data.frames with various column types.

The major functions is `merge`. It works like `cbind` or SQL’s `join`:

Let’s look at an example. It assumes that you are familiar with concepts of inner join, outer join, left join, and right join.

```
library(quantmod)
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
getSymbols("TWTR")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

[1] "TWTR"

```r
head(TWTR)
```

```
       TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2013-11-07 45.10 50.09 44.00 44.90 117701600 2013-11-08 45.93 46.94 40.69 41.65 27925300 2013-11-11 40.50 43.00 39.40 42.90 16113900 2013-11-12 43.66 43.78 41.83 41.90 6316700 2013-11-13 41.03 42.87 40.76 42.60 8688300 2013-11-14 42.34 45.67 42.24 44.69 11099400 TWTR.Adjusted 2013-11-07 44.90 2013-11-08 41.65 2013-11-11 42.90 2013-11-12 41.90 2013-11-13 42.60 2013-11-14 44.69

```r
str(TWTR)
```

An 'xts' object on 2013-11-07/2018-11-16 containing: Data: num [1:1267, 1:6] 45.1 45.9 40.5 43.7 41 ... - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:6] "TWTR.Open" "TWTR.High" "TWTR.Low" "TWTR.Close" ... Indexed by objects of class: [Date] TZ: UTC xts Attributes: List of 2 $ src : chr "yahoo" $ updated: POSIXct[1:1], format: "2018-11-17 10:28:26"

Note that the date obtained from the getSymbols function of the quantmod library is already an xts object. As such, we can merge it directly with our unemployment rate xts object constructed above. Nevertheless, it is instructive to examine the data using the View() function to ensure that you understand the number of observations resulting from the joined series.

```r
# 1. Inner join
TWTR_unemp01 <- merge(unemp_xts, TWTR, join = "inner")
  str(TWTR_unemp01)
```

An 'xts' object on 2014-04-01/2016-12-01 containing: Data: num [1:22, 1:7] 5.9 6.1 6.5 6.3 5.5 5.4 5.1 5.3 5.5 5.6 ... - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" ... Indexed by objects of class: [Date] TZ: UTC xts Attributes: NULL

```r
  head(TWTR_unemp01)
```

```
       unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2014-04-01 5.9 46.71 47.59 46.18 46.98 6916100 2014-05-01 6.1 39.01 40.77 38.97 39.09 15759800 2014-07-01 6.5 42.06 42.95 41.91 42.05 36019300 2014-08-01 6.3 45.01 45.54 43.81 44.13 37194800 2014-10-01 5.5 51.08 51.29 49.15 50.06 24733500 2014-12-01 5.4 41.29 41.29 39.00 39.04 22214000 TWTR.Adjusted 2014-04-01 46.98 2014-05-01 39.09 2014-07-01 42.05 2014-08-01 44.13 2014-10-01 50.06 2014-12-01 39.04

```r
# 2. Outer join (filling the missing observations with 99999)
# Basic argument use
TWTR_unemp02 <- merge(unemp_xts, TWTR, join = "outer", fill = 99999)
  str(TWTR_unemp02)
```

An 'xts' object on 2007-01-01/2018-11-16 containing: Data: num [1:1366, 1:7] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" … Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```r
  head(TWTR_unemp02)
```

```
       unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2007-01-01 5.0 99999 99999 99999 99999 99999 2007-02-01 4.9 99999 99999 99999 99999 99999 2007-03-01 4.5 99999 99999 99999 99999 99999 2007-04-01 4.3 99999 99999 99999 99999 99999 2007-05-01 4.3 99999 99999 99999 99999 99999 2007-06-01 4.7 99999 99999 99999 99999 99999 TWTR.Adjusted 2007-01-01 99999 2007-02-01 99999 2007-03-01 99999 2007-04-01 99999 2007-05-01 99999 2007-06-01 99999

```r
  #View(TWTR_unemp02)

# Left join
TWTR_unemp03 <- merge(unemp_xts, TWTR, join = "left", fill = 99999)
  str(TWTR_unemp03)
```

An 'xts' object on 2007-01-01/2017-01-01 containing: Data: num [1:121, 1:7] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" … Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```r
  head(TWTR_unemp03)
```

```
       unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2007-01-01 5.0 99999 99999 99999 99999 99999 2007-02-01 4.9 99999 99999 99999 99999 99999 2007-03-01 4.5 99999 99999 99999 99999 99999 2007-04-01 4.3 99999 99999 99999 99999 99999 2007-05-01 4.3 99999 99999 99999 99999 99999 2007-06-01 4.7 99999 99999 99999 99999 99999 TWTR.Adjusted 2007-01-01 99999 2007-02-01 99999 2007-03-01 99999 2007-04-01 99999 2007-05-01 99999 2007-06-01 99999

```r
  #View(TWTR_unemp03)

# Right join
TWTR_unemp04 <- merge(unemp_xts, TWTR, join = "right", fill = 99999)
  str(TWTR_unemp04)
```

An 'xts' object on 2013-11-07/2018-11-16 containing: Data: num [1:1267, 1:7] 99999 99999 99999 99999 99999 … - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:7] "unemp_xts" "TWTR.Open" "TWTR.High" "TWTR.Low" … Indexed by objects of class: [Date] TZ: UTC xts

Attributes:
NULL

```r
head(TWTR_unemp04)
```

```
unemp_xts TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2013-11-07 99999 45.10 50.09 44.00 44.90 117701600 2013-11-08 99999 45.93 46.94 40.69 41.65 27925300 2013-11-11 99999 40.50 43.00 39.40 42.90 16113900 2013-11-12 99999 43.66 43.78 41.83 41.90 6316700 2013-11-13 99999 41.03 42.87 40.76 42.60 8688300 2013-11-14 99999 42.34 45.67 42.24 44.69 11099400 TWTR.Adjusted 2013-11-07 44.90 2013-11-08 41.65 2013-11-11 42.90 2013-11-12 41.90 2013-11-13 42.60 2013-11-14 44.69

```r
#View(TWTR_unemp04)
```

Missing value imputation

xts also offers methods that allows filling missing values using last or previous observation. Note that I include this simply to point out that this is possible. I by no mean certify that this is the preferred method of imputing missing values in a time series. As I mentioned in live session, the specific method to use in missing value imputation is completely context dependent.

Filling missing values from the last observation

```r
# First, let's replace the "99999" values with NA and then exammine the series.

# Let's examine the first few dozen observations with NA
TWTR_unemp02['2013-10-01/2013-12-15'][,1]
```

```
unemp_xts
```

2013-10-01 7.0 2013-11-01 6.6 2013-11-07 99999.0 2013-11-08 99999.0 2013-11-11 99999.0 2013-11-12 99999.0 2013-11-13 99999.0 2013-11-14 99999.0 2013-11-15 99999.0 2013-11-18 99999.0 2013-11-19 99999.0 2013-11-20 99999.0 2013-11-21 99999.0 2013-11-22 99999.0 2013-11-25 99999.0 2013-11-26 99999.0 2013-11-27 99999.0 2013-11-29 99999.0 2013-12-01 6.5 2013-12-02 99999.0 2013-12-03 99999.0 2013-12-04 99999.0 2013-12-05 99999.0 2013-12-06 99999.0 2013-12-09 99999.0 2013-12-10 99999.0 2013-12-11 99999.0 2013-12-12 99999.0 2013-12-13 99999.0

```r
# Replace observations with "99999" with NA and store in a new series
unemp01 <- TWTR_unemp02[, 1]
unemp01['2013-10-01/2013-12-15']
```

```
unemp_xts
```

2013-10-01 7.0 2013-11-01 6.6 2013-11-07 99999.0 2013-11-08 99999.0 2013-11-11 99999.0 2013-11-12 99999.0 2013-11-13 99999.0 2013-11-14 99999.0 2013-11-15 99999.0 2013-11-18 99999.0 2013-11-19 99999.0 2013-11-20 99999.0 2013-11-21 99999.0 2013-11-22 99999.0 2013-11-25 99999.0 2013-11-26 99999.0 2013-11-27 99999.0 2013-11-29 99999.0 2013-12-01 6.5 2013-12-02 99999.0 2013-12-03 99999.0 2013-12-04 99999.0 2013-12-05 99999.0 2013-12-06 99999.0 2013-12-09 99999.0 2013-12-10 99999.0 2013-12-11 99999.0 2013-12-12 99999.0 2013-12-13 99999.0

```
str(unemp01)
```

An ‘xts’ object on 2007-01-01/2018-11-16 containing: Data: num [1:1366, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 … - attr(*, “dimnames”)=List of 2 ..$: NULL ..$: chr “unemp_xts” Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```
head(unemp01)
```

```
       unemp_xts
```

2007-01-01 5.0 2007-02-01 4.9 2007-03-01 4.5 2007-04-01 4.3 2007-05-01 4.3 2007-06-01 4.7

```
#TWTR_unemp02[, 1][TWTR_unemp02[, 1] >= 99990] <- NA

unemp02 <- unemp01
unemp02[unemp02 >= 99990] <- NA

cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'])
```

```
       unemp_xts unemp_xts.1
```

2013-10-01 7.0 7.0 2013-11-01 6.6 6.6 2013-11-07 99999.0 NA 2013-11-08 99999.0 NA 2013-11-11 99999.0 NA 2013-11-12 99999.0 NA 2013-11-13 99999.0 NA 2013-11-14 99999.0 NA 2013-11-15 99999.0 NA 2013-11-18 99999.0 NA 2013-11-19 99999.0 NA 2013-11-20 99999.0 NA 2013-11-21 99999.0 NA 2013-11-22 99999.0 NA 2013-11-25 99999.0 NA 2013-11-26 99999.0 NA 2013-11-27 99999.0 NA 2013-11-29 99999.0 NA 2013-12-01 6.5 6.5 2013-12-02 99999.0 NA 2013-12-03 99999.0 NA 2013-12-04 99999.0 NA 2013-12-05 99999.0 NA 2013-12-06 99999.0 NA 2013-12-09 99999.0 NA 2013-12-10 99999.0 NA 2013-12-11 99999.0 NA 2013-12-12 99999.0 NA 2013-12-13 99999.0 NA

```
# Impute the missing values (stored as NA) with the last observation
TWTR_unemp02_v2a <- na.locf(TWTR_unemp02[,1],
                            na.rm = TRUE, fromLast = TRUE)
unemp03 <- unemp02
unemp03 <- na.locf(unemp03, na.rm = TRUE, fromLast = FALSE);

# Examine the pre- and post-imputed series
cbind(TWTR_unemp02['2013-10-01/2013-12-30'][,1], TWTR_unemp02_v2a['2013-10-01/2013-12-15'])
```

```
       unemp_xts unemp_xts.1
```

2013-10-01 7.0 7.0 2013-11-01 6.6 6.6 2013-11-07 99999.0 99999.0 2013-11-08 99999.0 99999.0 2013-11-11 99999.0 99999.0 2013-11-12 99999.0 99999.0 2013-11-13 99999.0 99999.0 2013-11-14 99999.0 99999.0 2013-11-15 99999.0 99999.0 2013-11-18 99999.0 99999.0 2013-11-19 99999.0 99999.0 2013-11-20 99999.0 99999.0 2013-11-21 99999.0 99999.0 2013-11-22 99999.0 99999.0 2013-11-25 99999.0 99999.0 2013-11-26 99999.0 99999.0 2013-11-27 99999.0 99999.0 2013-11-29 99999.0 99999.0 2013-12-01 6.5 6.5 2013-12-02 99999.0 99999.0 2013-12-03 99999.0 99999.0 2013-12-04 99999.0 99999.0 2013-12-05 99999.0 99999.0 2013-12-06 99999.0 99999.0 2013-12-09 99999.0 99999.0 2013-12-10 99999.0 99999.0 2013-12-11 99999.0 99999.0 2013-12-12 99999.0 99999.0 2013-12-13 99999.0 99999.0 2013-12-16 99999.0 NA 2013-12-17 99999.0 NA 2013-12-18 99999.0 NA 2013-12-19 99999.0 NA 2013-12-20 99999.0 NA 2013-12-23 99999.0 NA 2013-12-24 99999.0 NA 2013-12-26 99999.0 NA 2013-12-27

99999.0 NA 2013-12-30 99999.0 NA

```r
cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'],
unemp03['2013-10-01/2013-12-15'])
```

```
       unemp_xts unemp_xts.1 unemp_xts.2
```

2013-10-01 7.0 7.0 7.0 2013-11-01 6.6 6.6 6.6 2013-11-07 99999.0 NA 6.6 2013-11-08 99999.0 NA 6.6 2013-11-11 99999.0 NA 6.6 2013-11-12 99999.0 NA 6.6 2013-11-13 99999.0 NA 6.6 2013-11-14 99999.0 NA 6.6 2013-11-15 99999.0 NA 6.6 2013-11-18 99999.0 NA 6.6 2013-11-19 99999.0 NA 6.6 2013-11-20 99999.0 NA 6.6 2013-11-21 99999.0 NA 6.6 2013-11-22 99999.0 NA 6.6 2013-11-25 99999.0 NA 6.6 2013-11-26 99999.0 NA 6.6 2013-11-27 99999.0 NA 6.6 2013-11-29 99999.0 NA 6.6 2013-12-01 6.5 6.5 6.5 2013-12-02 99999.0 NA 6.5 2013-12-03 99999.0 NA 6.5 2013-12-04 99999.0 NA 6.5 2013-12-05 99999.0 NA 6.5 2013-12-06 99999.0 NA 6.5 2013-12-09 99999.0 NA 6.5 2013-12-10 99999.0 NA 6.5 2013-12-11 99999.0 NA 6.5 2013-12-12 99999.0 NA 6.5 2013-12-13 99999.0 NA 6.5

Another missing value imputation method is linear interpolation, which can also be easily done in xts objects. In the following example, we use linear interpolation to fill in the NA in between months. The result is stored in `unemp04`. Note in the following the different ways of imputing missing values.

```r
unemp04 <- unemp02
unemp04['2013-10-01/2014-02-01']
```

```
       unemp_xts
```

2013-10-01 7.0 2013-11-01 6.6 2013-11-07 NA 2013-11-08 NA 2013-11-11 NA 2013-11-12 NA 2013-11-13 NA 2013-11-14 NA 2013-11-15 NA 2013-11-18 NA 2013-11-19 NA 2013-11-20 NA 2013-11-21 NA 2013-11-22 NA 2013-11-25 NA 2013-11-26 NA 2013-11-27 NA 2013-11-29 NA 2013-12-01 6.5 2013-12-02 NA 2013-12-03 NA 2013-12-04 NA 2013-12-05 NA 2013-12-06 NA 2013-12-09 NA 2013-12-10 NA 2013-12-11 NA 2013-12-12 NA 2013-12-13 NA 2013-12-16 NA 2013-12-17 NA 2013-12-18 NA 2013-12-19 NA 2013-12-20 NA 2013-12-23 NA 2013-12-24 NA 2013-12-26 NA 2013-12-27 NA 2013-12-30 NA 2013-12-31 NA 2014-01-01 7.0 2014-01-02 NA 2014-01-03 NA 2014-01-06 NA 2014-01-07 NA 2014-01-08 NA 2014-01-09 NA 2014-01-10 NA 2014-01-13 NA 2014-01-14 NA 2014-01-15 NA 2014-01-16 NA 2014-01-17 NA 2014-01-21 NA 2014-01-22 NA 2014-01-23 NA 2014-01-24 NA 2014-01-27 NA 2014-01-28 NA 2014-01-29 NA 2014-01-30 NA 2014-01-31 NA 2014-02-01 7.0

```r
unemp04 <- na.approx(unemp04, maxgap=31)
unemp04['2013-10-01/2014-02-01']
```

```
       unemp_xts
```

2013-10-01 7.000000 2013-11-01 6.600000 2013-11-07 6.580000 2013-11-08 6.576667 2013-11-11 6.566667 2013-11-12 6.563333 2013-11-13 6.560000 2013-11-14 6.556667 2013-11-15 6.553333 2013-11-18 6.543333 2013-11-19 6.540000 2013-11-20 6.536667 2013-11-21 6.533333 2013-11-22 6.530000 2013-11-25 6.520000 2013-11-26 6.516667 2013-11-27 6.513333 2013-11-29 6.506667 2013-12-01 6.500000 2013-12-02 6.516129 2013-12-03 6.532258 2013-12-04 6.548387 2013-12-05 6.564516 2013-12-06 6.580645 2013-12-09 6.629032 2013-12-10 6.645161 2013-12-11 6.661290 2013-12-12 6.677419 2013-12-13 6.693548 2013-12-16 6.741935 2013-12-17 6.758065 2013-12-18 6.774194 2013-12-19 6.790323 2013-12-20 6.806452 2013-12-23 6.854839 2013-12-24 6.870968 2013-12-26 6.903226 2013-12-27 6.919355 2013-12-30 6.967742 2013-12-31 6.983871 2014-01-01 7.000000 2014-01-02 7.000000 2014-01-03 7.000000 2014-01-06 7.000000 2014-01-07 7.000000

2014-01-08 7.000000 2014-01-09 7.000000 2014-01-10 7.000000 2014-01-13 7.000000 2014-01-14 7.000000 2014-01-15 7.000000 2014-01-16 7.000000 2014-01-17 7.000000 2014-01-21 7.000000 2014-01-22 7.000000 2014-01-23 7.000000 2014-01-24 7.000000 2014-01-27 7.000000 2014-01-28 7.000000 2014-01-29 7.000000 2014-01-30 7.000000 2014-01-31 7.000000 2014-02-01 7.000000

```
round(cbind(unemp01['2013-10-01/2013-12-15'], unemp02['2013-10-01/2013-12-15'],
unemp03['2013-10-01/2013-12-15'],
unemp04['2013-10-01/2013-12-15']),2)
```

```
        unemp_xts unemp_xts.1 unemp_xts.2 unemp_xts.3
```

2013-10-01 7.0 7.0 7.0 7.00 2013-11-01 6.6 6.6 6.6 6.60 2013-11-07 99999.0 NA 6.6 6.58 2013-11-08 99999.0 NA 6.6 6.58 2013-11-11 99999.0 NA 6.6 6.57 2013-11-12 99999.0 NA 6.6 6.56 2013-11-13 99999.0 NA 6.6 6.56 2013-11-14 99999.0 NA 6.6 6.56 2013-11-15 99999.0 NA 6.6 6.55 2013-11-18 99999.0 NA 6.6 6.54 2013-11-19 99999.0 NA 6.6 6.54 2013-11-20 99999.0 NA 6.6 6.54 2013-11-21 99999.0 NA 6.6 6.53 2013-11-22 99999.0 NA 6.6 6.53 2013-11-25 99999.0 NA 6.6 6.52 2013-11-26 99999.0 NA 6.6 6.52 2013-11-27 99999.0 NA 6.6 6.51 2013-11-29 99999.0 NA 6.6 6.51 2013-12-01 6.5 6.5 6.5 6.50 2013-12-02 99999.0 NA 6.5 6.52 2013-12-03 99999.0 NA 6.5 6.53 2013-12-04 99999.0 NA 6.5 6.55 2013-12-05 99999.0 NA 6.5 6.56 2013-12-06 99999.0 NA 6.5 6.58 2013-12-09 99999.0 NA 6.5 6.63 2013-12-10 99999.0 NA 6.5 6.65 2013-12-11 99999.0 NA 6.5 6.66 2013-12-12 99999.0 NA 6.5 6.68 2013-12-13 99999.0 NA 6.5 6.69

Calculate difference in time series

A very common operation on time series is to take a difference of the series to transform a non-stationary serier to a stationary series. First order differencing takes the form $x(t) - x(t - k)$ where k denotes the number of time lags. Higher order differences are simply the reapplication of a difference to each prior result (like a second derivative or a difference of the difference).

Let's use the `unemp_xts` series as examples:

```
str(unemp_xts)
```

An 'xts' object on 2007-01-01/2017-01-01 containing: Data: num [1:121, 1] 5 4.9 4.5 4.3 4.3 4.7 4.9 4.6 4.5 4.4 ... Indexed by objects of class: [Date] TZ: UTC xts Attributes:
NULL

```
unemp_xts
```

```
        [,1]
```

2007-01-01 5.0 2007-02-01 4.9 2007-03-01 4.5 2007-04-01 4.3 2007-05-01 4.3 2007-06-01 4.7 2007-07-01 4.9 2007-08-01 4.6 2007-09-01 4.5 2007-10-01 4.4 2007-11-01 4.5 2007-12-01 4.8 2008-01-01 5.4 2008-02-01 5.2 2008-03-01 5.2 2008-04-01 4.8 2008-05-01 5.2 2008-06-01 5.7 2008-07-01 6.0 2008-08-01 6.1 2008-09-01 6.0 2008-10-01 6.1 2008-11-01 6.5 2008-12-01 7.1 2009-01-01 8.5 2009-02-01 8.9 2009-03-01 9.0 2009-04-01 8.6 2009-05-01 9.1 2009-06-01 9.7 2009-07-01 9.7 2009-08-01 9.6 2009-09-01 9.5 2009-10-01 9.5 2009-11-01 9.4 2009-12-01 9.7 2010-01-01 10.6 2010-02-01 10.4 2010-03-01 10.2 2010-04-01 9.5 2010-05-01 9.3 2010-06-01 9.6 2010-07-01 9.7 2010-08-01 9.5 2010-09-01 9.2 2010-10-01 9.0 2010-11-01 9.3 2010-12-01 9.1 2011-01-01 9.8 2011-02-01 9.5 2011-03-01 9.2 2011-04-01 8.7 2011-05-01 8.7 2011-06-01 9.3 2011-07-01 9.3 2011-08-01 9.1 2011-09-01 8.8 2011-10-01 8.5 2011-11-01 8.2 2011-12-01 8.3 2012-01-01 8.8 2012-02-01 8.7 2012-03-01 8.4 2012-04-01 7.7 2012-05-01 7.9

2012-06-01 8.4 2012-07-01 8.6 2012-08-01 8.2 2012-09-01 7.6 2012-10-01 7.5 2012-11-01 7.4 2012-12-01 7.6 2013-01-01 8.5 2013-02-01 8.1 2013-03-01 7.6 2013-04-01 7.1 2013-05-01 7.3 2013-06-01 7.8 2013-07-01 7.7 2013-08-01 7.3 2013-09-01 7.0 2013-10-01 7.0 2013-11-01 6.6 2013-12-01 6.5 2014-01-01 7.0 2014-02-01 7.0 2014-03-01 6.8 2014-04-01 5.9 2014-05-01 6.1 2014-06-01 6.3 2014-07-01 6.5 2014-08-01 6.3 2014-09-01 5.7 2014-10-01 5.5 2014-11-01 5.5 2014-12-01 5.4 2015-01-01 6.1 2015-02-01 5.8 2015-03-01 5.6 2015-04-01 5.1 2015-05-01 5.3 2015-06-01 5.5 2015-07-01 5.6 2015-08-01 5.2 2015-09-01 4.9 2015-10-01 4.8 2015-11-01 4.8 2015-12-01 4.8 2016-01-01 5.3 2016-02-01 5.2 2016-03-01 5.1 2016-04-01 4.7 2016-05-01 4.5 2016-06-01 5.1 2016-07-01 5.1 2016-08-01 5.0 2016-09-01 4.8 2016-10-01 4.7 2016-11-01 4.4 2016-12-01 4.5 2017-01-01 5.1

```r
diff(unemp_xts, lag = 1, difference = 1, log = FALSE, na.pad = TRUE)
```

```
        [,1]
```

2007-01-01 NA 2007-02-01 -0.1 2007-03-01 -0.4 2007-04-01 -0.2 2007-05-01 0.0 2007-06-01 0.4 2007-07-01 0.2 2007-08-01 -0.3 2007-09-01 -0.1 2007-10-01 -0.1 2007-11-01 0.1 2007-12-01 0.3 2008-01-01 0.6 2008-02-01 -0.2 2008-03-01 0.0 2008-04-01 -0.4 2008-05-01 0.4 2008-06-01 0.5 2008-07-01 0.3 2008-08-01 0.1 2008-09-01 -0.1 2008-10-01 0.1 2008-11-01 0.4 2008-12-01 0.6 2009-01-01 1.4 2009-02-01 0.4 2009-03-01 0.1 2009-04-01 -0.4 2009-05-01 0.5 2009-06-01 0.6 2009-07-01 0.0 2009-08-01 -0.1 2009-09-01 -0.1 2009-10-01 0.0 2009-11-01 -0.1 2009-12-01 0.3 2010-01-01 0.9 2010-02-01 -0.2 2010-03-01 -0.2 2010-04-01 -0.7 2010-05-01 -0.2 2010-06-01 0.3 2010-07-01 0.1 2010-08-01 -0.2 2010-09-01 -0.3 2010-10-01 -0.2 2010-11-01 0.3 2010-12-01 -0.2 2011-01-01 0.7 2011-02-01 -0.3 2011-03-01 -0.3 2011-04-01 -0.5 2011-05-01 0.0 2011-06-01 0.6 2011-07-01 0.0 2011-08-01 -0.2 2011-09-01 -0.3 2011-10-01 -0.3 2011-11-01 -0.3 2011-12-01 0.1 2012-01-01 0.5 2012-02-01 -0.1 2012-03-01 -0.3 2012-04-01 -0.7 2012-05-01 0.2 2012-06-01 0.5 2012-07-01 0.2 2012-08-01 -0.4 2012-09-01 -0.6 2012-10-01 -0.1 2012-11-01 -0.1 2012-12-01 0.2 2013-01-01 0.9 2013-02-01 -0.4 2013-03-01 -0.5 2013-04-01 -0.5 2013-05-01 0.2 2013-06-01 0.5 2013-07-01 -0.1 2013-08-01 -0.4 2013-09-01 -0.3 2013-10-01 0.0 2013-11-01 -0.4 2013-12-01 -0.1 2014-01-01 0.5 2014-02-01 0.0 2014-03-01 -0.2 2014-04-01 -0.9 2014-05-01 0.2 2014-06-01 0.2 2014-07-01 0.2 2014-08-01 -0.2 2014-09-01 -0.6 2014-10-01 -0.2 2014-11-01 0.0 2014-12-01 -0.1 2015-01-01 0.7 2015-02-01 -0.3 2015-03-01 -0.2 2015-04-01 -0.5 2015-05-01 0.2 2015-06-01 0.2 2015-07-01 0.1 2015-08-01 -0.4 2015-09-01 -0.3 2015-10-01 -0.1 2015-11-01 0.0 2015-12-01 0.0 2016-01-01 0.5 2016-02-01 -0.1 2016-03-01 -0.1 2016-04-01 -0.4 2016-05-01 -0.2 2016-06-01 0.6 2016-07-01 0.0 2016-08-01 -0.1 2016-09-01 -0.2 2016-10-01 -0.1 2016-11-01 -0.3 2016-12-01 0.1 2017-01-01 0.6

```r
# calculate the first difference of AirPass using lag and subtraction
#AirPass - lag(AirPass, k = 1)

# calculate the first order 12-month difference if AirPass
diff(unemp_xts, lag = 12, differences = 1)
```

```
        [,1]
```

2007-01-01 NA 2007-02-01 NA 2007-03-01 NA 2007-04-01 NA 2007-05-01 NA 2007-06-01 NA 2007-07-01 NA 2007-08-01 NA 2007-09-01 NA 2007-10-01 NA 2007-11-01 NA 2007-12-01 NA 2008-01-01 0.4 2008-02-01 0.3 2008-03-01 0.7 2008-04-01 0.5 2008-05-01 0.9 2008-06-01 1.0 2008-07-01 1.1 2008-08-01 1.5 2008-09-01 1.5 2008-10-01 1.7 2008-11-01 2.0 2008-12-01 2.3 2009-01-01 3.1 2009-02-01 3.7 2009-03-01 3.8 2009-04-01 3.8 2009-05-01 3.9 2009-06-01 4.0 2009-07-01 3.7 2009-08-01 3.5 2009-09-01 3.5 2009-10-01 3.4 2009-11-01 2.9 2009-12-01 2.6 2010-01-01 2.1 2010-02-01 1.5 2010-03-01 1.2 2010-04-01 0.9 2010-05-01 0.2 2010-06-01 -0.1 2010-07-01 0.0 2010-08-01 -0.1 2010-09-01 -0.3 2010-

10-01 -0.5 2010-11-01 -0.1 2010-12-01 -0.6 2011-01-01 -0.8 2011-02-01 -0.9 2011-03-01 -1.0 2011-04-01 -0.8 2011-05-01 -0.6 2011-06-01 -0.3 2011-07-01 -0.4 2011-08-01 -0.4 2011-09-01 -0.4 2011-10-01 -0.5 2011-11-01 -1.1 2011-12-01 -0.8 2012-01-01 -1.0 2012-02-01 -0.8 2012-03-01 -0.8 2012-04-01 -1.0 2012-05-01 -0.8 2012-06-01 -0.9 2012-07-01 -0.7 2012-08-01 -0.9 2012-09-01 -1.2 2012-10-01 -1.0 2012-11-01 -0.8 2012-12-01 -0.7 2013-01-01 -0.3 2013-02-01 -0.6 2013-03-01 -0.8 2013-04-01 -0.6 2013-05-01 -0.6 2013-06-01 -0.6 2013-07-01 -0.9 2013-08-01 -0.9 2013-09-01 -0.6 2013-10-01 -0.5 2013-11-01 -0.8 2013-12-01 -1.1 2014-01-01 -1.5 2014-02-01 -1.1 2014-03-01 -0.8 2014-04-01 -1.2 2014-05-01 -1.2 2014-06-01 -1.5 2014-07-01 -1.2 2014-08-01 -1.0 2014-09-01 -1.3 2014-10-01 -1.5 2014-11-01 -1.1 2014-12-01 -1.1 2015-01-01 -0.9 2015-02-01 -1.2 2015-03-01 -1.2 2015-04-01 -0.8 2015-05-01 -0.8 2015-06-01 -0.8 2015-07-01 -0.9 2015-08-01 -1.1 2015-09-01 -0.8 2015-10-01 -0.7 2015-11-01 -0.7 2015-12-01 -0.6 2016-01-01 -0.8 2016-02-01 -0.6 2016-03-01 -0.5 2016-04-01 -0.4 2016-05-01 -0.8 2016-06-01 -0.4 2016-07-01 -0.5 2016-08-01 -0.2 2016-09-01 -0.1 2016-10-01 -0.1 2016-11-01 -0.4 2016-12-01 -0.3 2017-01-01 -0.2

Task 4:

1. Read A. Section 3.4 of "xts: Extensible Time Series" by Jeffrey A. Ryan and Joshua M. Ulrich

   B. the following questions in "xts FAQ" a. I am using apply() to run a custom function on my xts series. Why the returned matrix has diâ µerent dimensions than the original one?

2. Follow the following two sections of this document

Apply various functions to time series

The family of `apply` functions perhaps is one of the most powerful R function families. In time series, `xts` provides `period.apply`, which takes (1) a time series, (2) an index of endpoints, and (3) a function to apply. It takes the following general form:

`period.apply(x, INDEX, FUN, ...)`

As an example, we use the Twitter stock price series (to be precise, the daily closing price), create an index storing the points corresopnding to the weeks of the daily series, and apply functions to calculate the weekly mean.

```
# Step 1: Identify the endpoints; in this case, we use weekly time interval. That is, we extra

#View(TWTR)
head(TWTR)
```

```
       TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
```

2013-11-07 45.10 50.09 44.00 44.90 117701600 2013-11-08 45.93 46.94 40.69 41.65 27925300 2013-11-11 40.50 43.00 39.40 42.90 16113900 2013-11-12 43.66 43.78 41.83 41.90 6316700 2013-11-13 41.03 42.87 40.76 42.60 8688300 2013-11-14 42.34 45.67 42.24 44.69 11099400 TWTR.Adjusted 2013-11-07 44.90 2013-11-08 41.65 2013-11-11 42.90 2013-11-12 41.90 2013-11-13 42.60 2013-11-14 44.69

```
TWTR_ep <- endpoints(TWTR[,4], on = "weeks")
#TWTR_ep

# Step 2: Calculate the weekly mean
```

```r
TWTR.Close_weeklyMean <- period.apply(TWTR[, 4], INDEX = TWTR_ep, FUN = mean)
head(round(TWTR.Close_weeklyMean,2),8)
```

```
       TWTR.Close
```

2013-11-08 43.28 2013-11-15 43.21 2013-11-22 41.40 2013-11-29 40.43 2013-12-06 43.28 2013-12-13 53.56 2013-12-20 57.21 2013-12-27 67.89

The power of the apply function really comes with the use of custom-defined function. For instance, we can easily

```r
f <- function(x) {
  mean <- mean(x)
  quantile <- quantile(x,c(0.05,0.25,0.50,0.75,0.95))
  sd <- sd(x)

  result <- c(mean, sd, quantile)
  return(result)
}
head(round(period.apply(TWTR[, 4], INDEX = TWTR_ep, FUN = f),2),10)
```

```
                     5%    25%    50%    75%    95%
```

2013-11-08 43.28 2.30 41.81 42.46 43.28 44.09 44.74 2013-11-15 43.21 1.11 42.04 42.60 42.90 43.98 44.55 2013-11-22 41.40 0.48 41.01 41.05 41.14 41.75 42.00 2013-11-29 40.43 1.07 39.23 39.90 40.54 41.07 41.47 2013-12-06 43.28 2.14 40.90 41.37 43.69 44.95 45.49 2013-12-13 53.56 3.75 49.71 51.99 52.34 55.33 58.27 2013-12-20 57.21 1.71 55.70 56.45 56.61 57.49 59.51 2013-12-27 67.89 4.55 63.87 64.34 67.25 70.80 72.81 2014-01-03 65.17 3.84 60.98 62.87 65.58 67.88 68.78 2014-01-10 60.22 3.86 57.01 57.05 59.29 61.46 65.32

Calculate basic rolling statistics of series by month

Using `rollapply`, one can calculate rolling statistics of a series:

```r
# Calculate rolling mean over a 10-day period and print it with the original series
head(cbind(TWTR[,4], rollapply(TWTR[, 4], 10, FUN = mean, na.rm = TRUE)),15)
```

```
       TWTR.Close TWTR.Close.1
```

2013-11-07 44.90 NA 2013-11-08 41.65 NA 2013-11-11 42.90 NA 2013-11-12 41.90 NA 2013-11-13 42.60 NA 2013-11-14 44.69 NA 2013-11-15 43.98 NA 2013-11-18 41.14 NA 2013-11-19 41.75 NA 2013-11-20 41.05 42.656 2013-11-21 42.06 42.372 2013-11-22 41.00 42.307 2013-11-25 39.06 41.923 2013-11-26 40.18 41.751 2013-11-27 40.90 41.581

Task 5:

1. Read AMAZ.csv and UMCSENT.csv into R as R DataFrames

```r
amaz=read.csv('AMAZ.csv')
umcsent=read.csv('UMCSENT.csv')
head(amaz)
```

```
   Index AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
```

1 2007-01-03 20.0 20.0 16.0 16.0 650 2 2007-01-04 20.0 20.0 20.0 20.0 67 3 2007-01-08 19.2 22.0 19.2 22.0 1801 4 2007-01-09 22.0 22.0 20.8 20.8 356 5 2007-01-10 20.8 20.8 20.8 20.8 438 6 2007-01-11 20.8 21.6 20.8 21.6 2318

```r
tail(amaz)
```

```
      Index AMAZ.Open AMAZ.High AMAZ.Low AMAZ.Close AMAZ.Volume
```

1174 2013-01-04 0.88 0.88 0.80 0.80 3850 1175 2013-01-07 0.80 1.00 0.80 1.00 2715 1176 2013-01-08 0.80 0.80 0.68 0.68 4668 1177 2013-01-09 0.88 0.88 0.80 0.80 2750 1178 2013-01-11 0.80 0.80 0.80 0.80 3000 1179 2013-01-15 0.68 0.68 0.68 0.68 1000

2. Convert them to xts objects

```r
#amaz.xts=as.xts(amaz[,2:6],Index=as.Date(amaz$Index,"%Y-%m-%d"))
```

3. Merge the two set of series together, perserving all of the obserbvations in both set of series
   a. fill all of the missing values of the UMCSENT series with -9999
   b. then create a new series, named UMCSENT02, from the original UMCSENT series replace all of the -9999 with NAs
   c. then create a new series, named UMCSENT03, and replace the NAs with the last observation
   d. then create a new series, named UMCSENT04, and replace the NAs using linear interpolation.
   e. Print out some observations to ensure that your merge as well as the missing value imputation are done correctly. I leave it up to you to decide exactly how many observations to print; do something that makes sense. (Hint: Do not print out the entire dataset!)
4. Calculate the daily return of the Amazon closing price (AMAZ.close), where daily return is defined as $(x(t) - x(t-1))/x(t-1)$. Plot the daily return series.
5. Create a 20-day and a 50-day rolling mean series from the AMAZ.close series.