Exam 3

Michael Pena

2024-06-13

Question 1

We found previously that

$$\nabla_4 X_t = 4\beta_1 + Z_t + Z_{t-4}$$

for the seasonal differencing, we find that

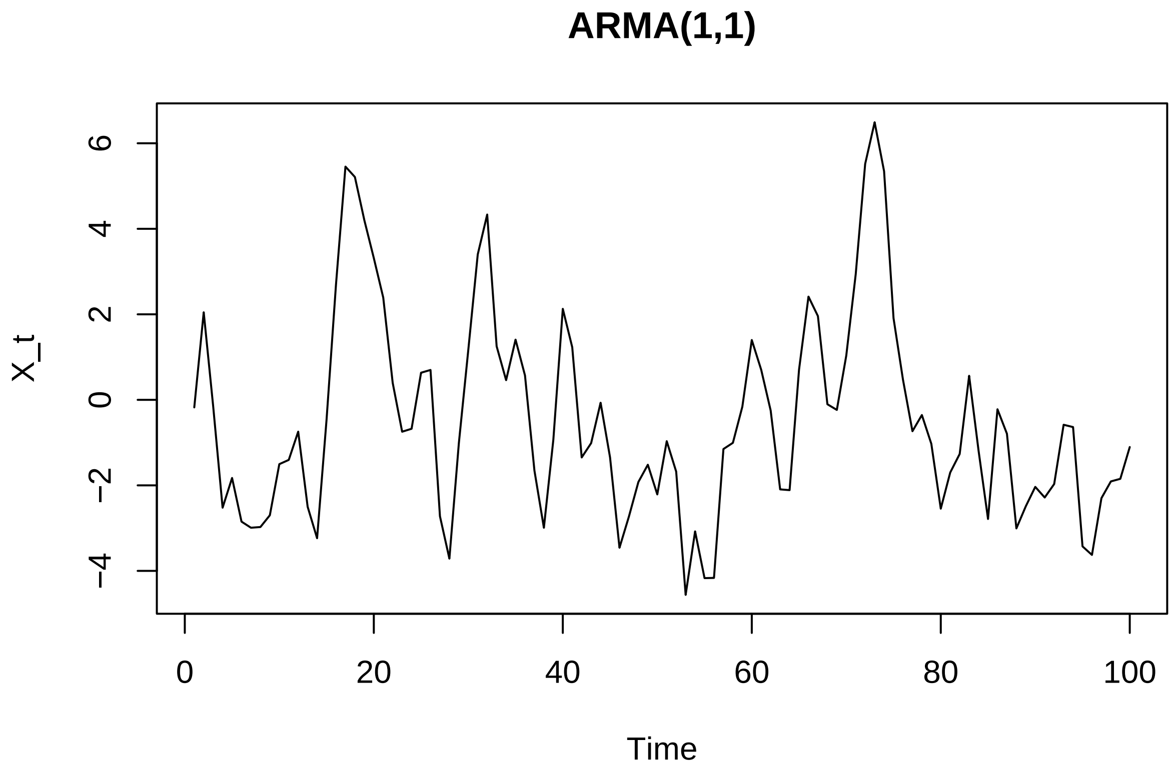
$$\nabla_4 \nabla X_t = -Z_{t-1} - Z_{t-4} + Z_{t-5}$$

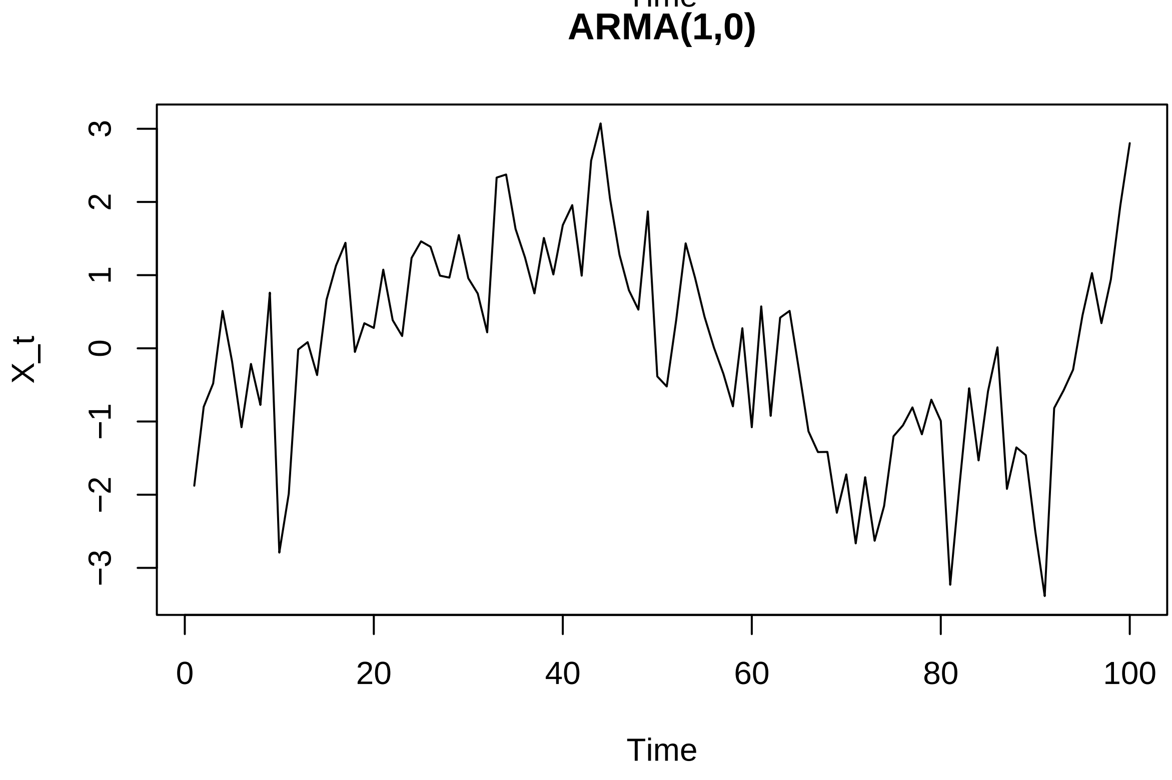
One has more parameters than the other. essentially in the end we want to go with the model that works with less parameters and does the same job in the end; less parameters means less work for the same result (we would pick the seasonal differenced pathway).

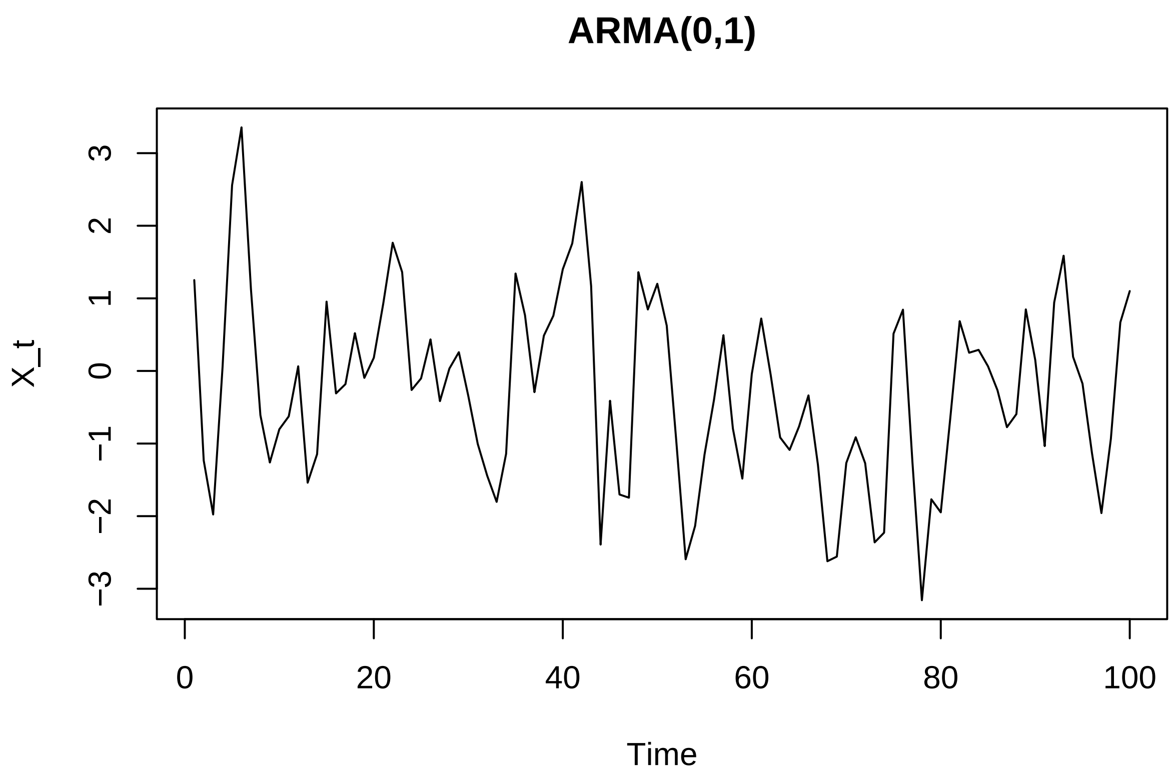
Question 2

```
# define the coeffs
phi = 0.6
the = 0.9
# save simulations into a matrix
SIM  <- matrix(0,100,3)
SIM[,1] <- arima.sim(n=100, list(ar = phi, ma = the))
SIM[,2] <- arima.sim(n=100, list(ar = phi))
SIM[,3] <- arima.sim(n=100, list(ma = the))
```

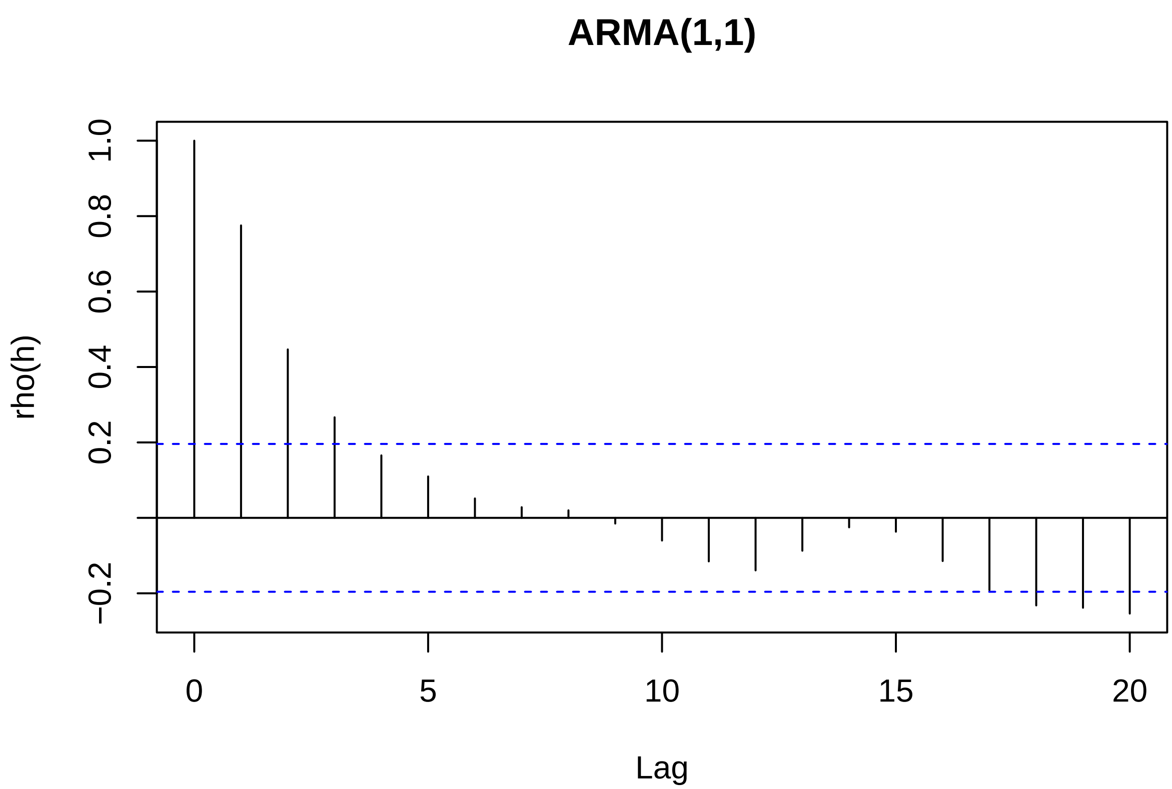
```
# plot the timeseries
name = c("ARMA(1,1)","ARMA(1,0)","ARMA(0,1)")
for(i in 1:3){plot.ts(SIM[,i],main = name[i], ylab = "X_t")}
```

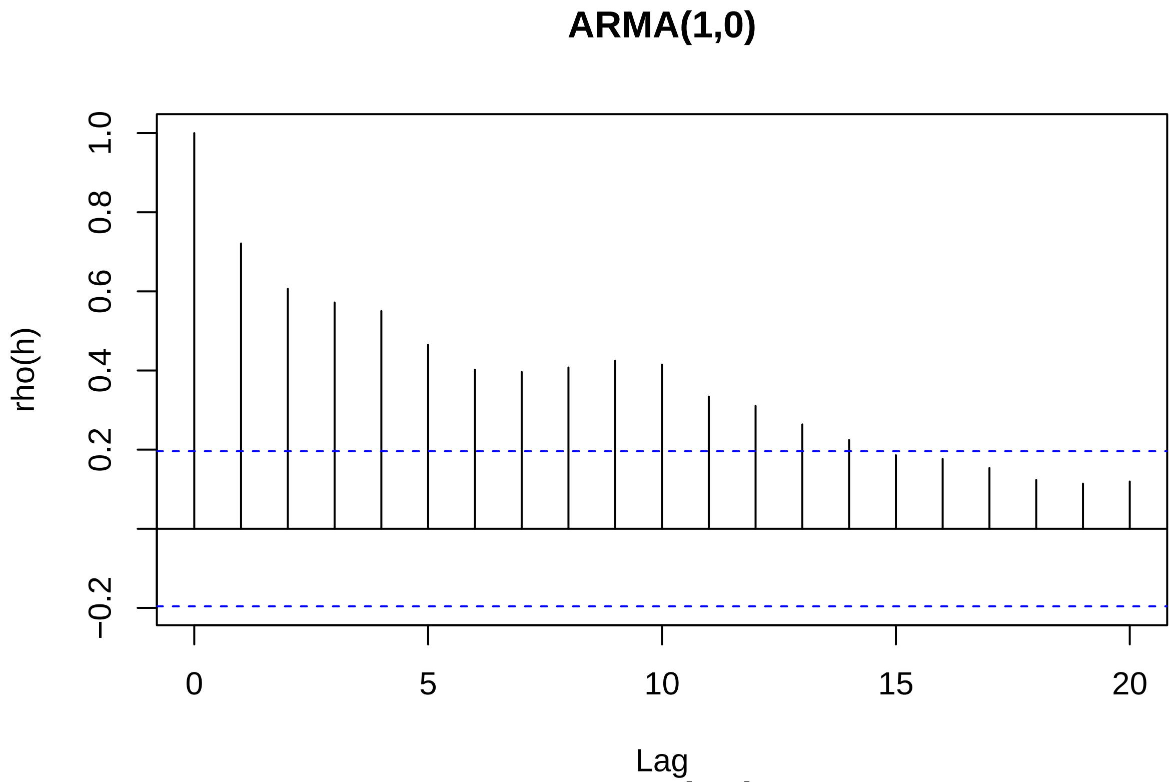

ARMA(1,1)
X_t
6
4
2
0
−2
−4
0
20
40
60
80
100
Time

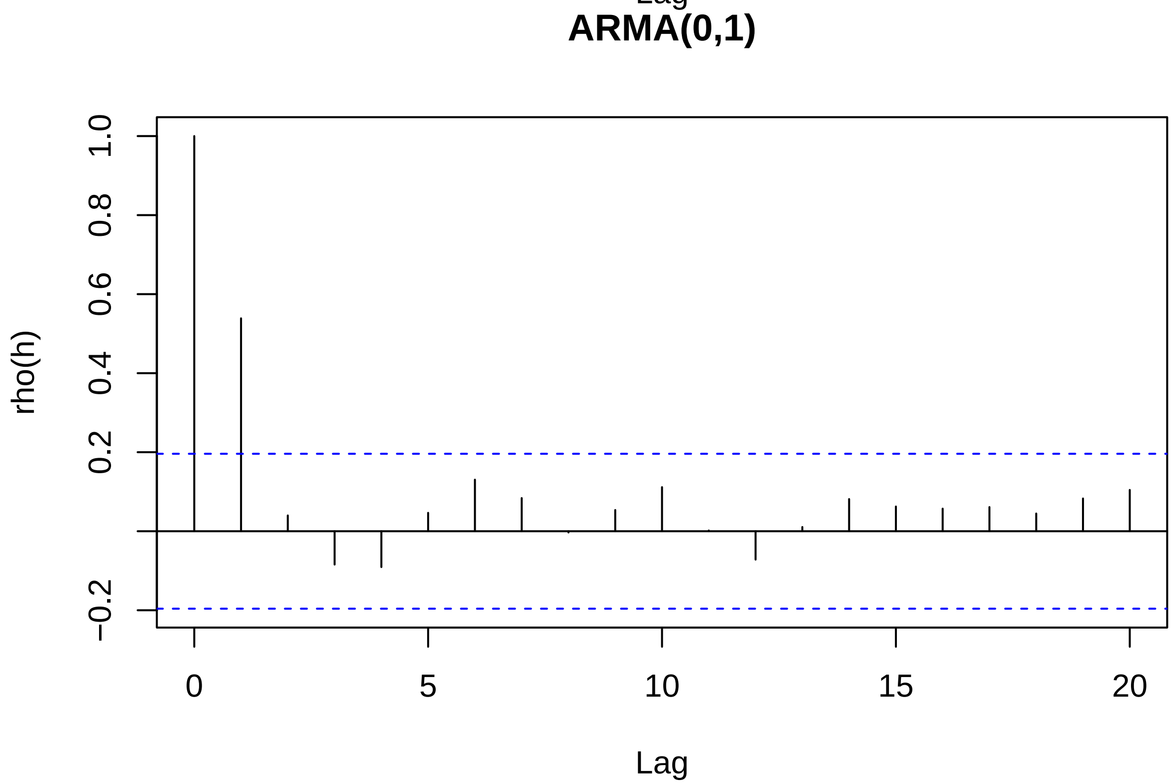

ARMA(1,0)
X_t
3
2
1
0
−1
−2
−3
0
20
40
60
80
100
Time



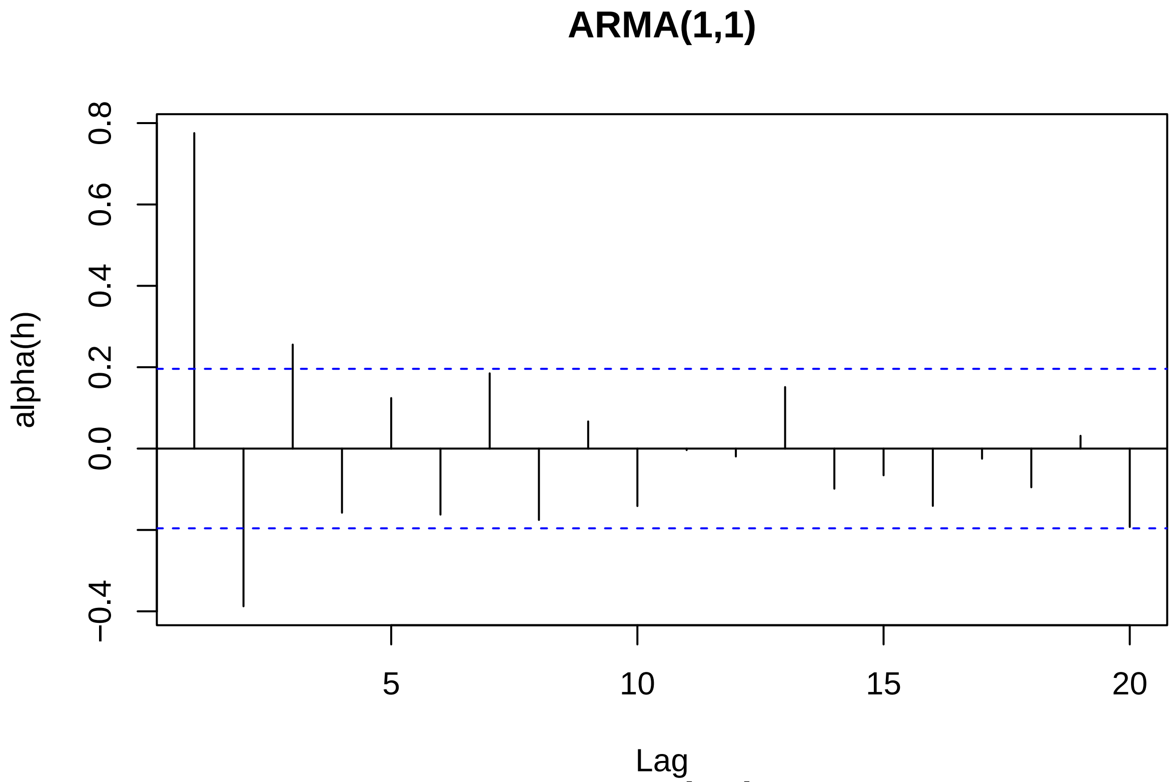

```r
# plot the ACFs
for(i in 1:3){acf(SIM[,i],main = name[i],ylab = "rho(h)")}
```

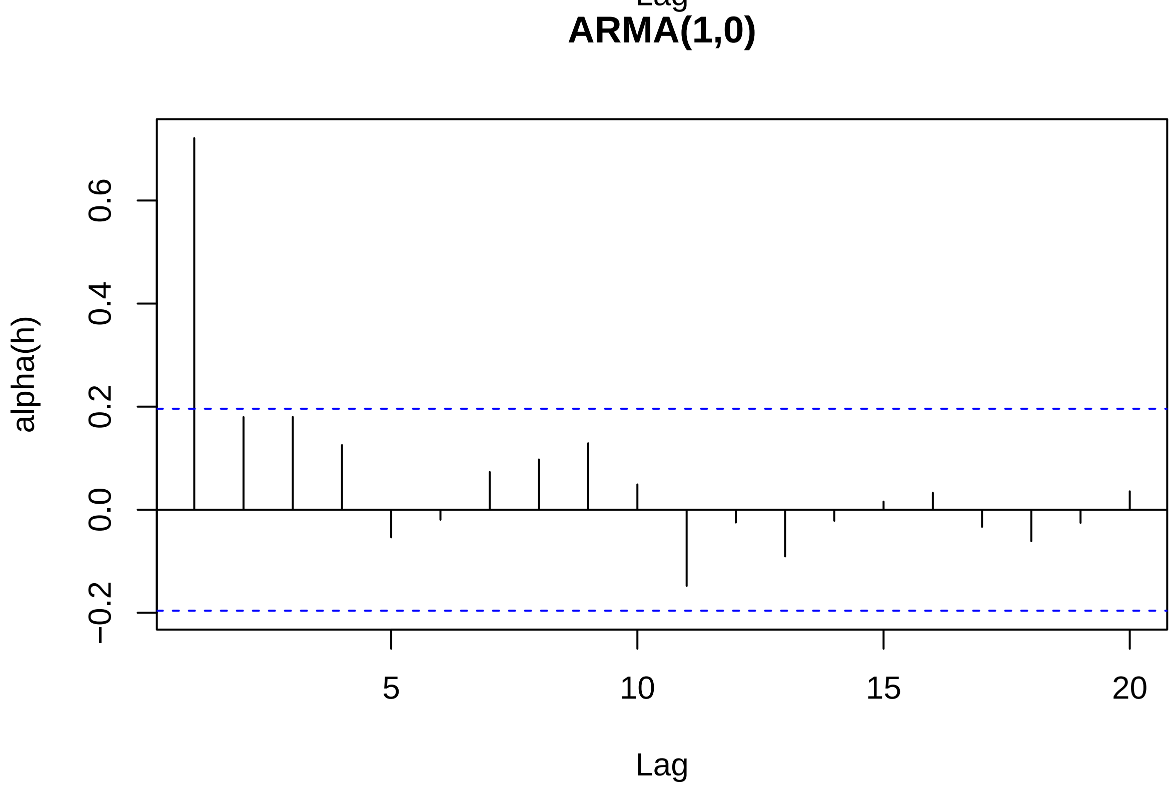


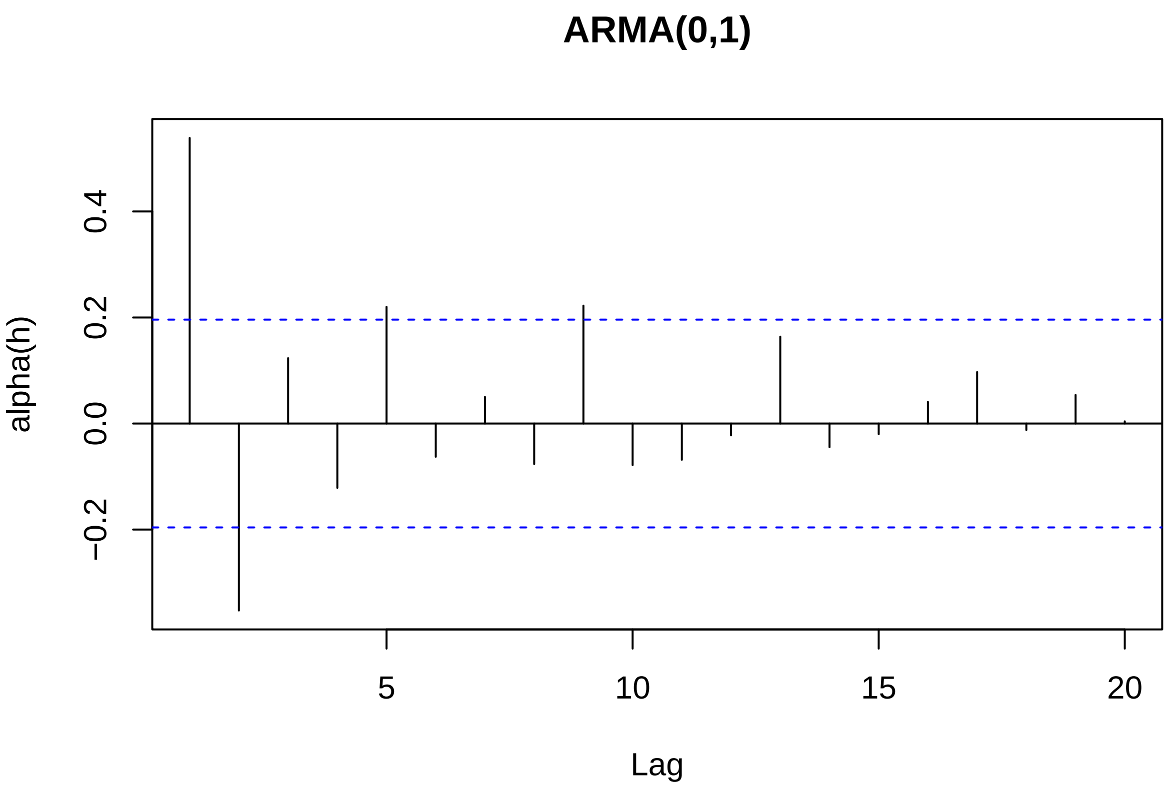





```r
# plot the PACFs
for(i in 1:3){pacf(SIM[,i],main = name[i],ylab = "alpha(h)")}
```


ARMA(1,1)
alpha(h)
0.8
0.6
0.4
0.2
0.0
−0.4
5
10
15
20
Lag


ARMA(1,0)
alpha(h)
0.6
0.4
0.2
0.0
−0.2
5
10
15
20
Lag



The models seem to match the table we went over in class; I do think that even though there seems to be a significant peak in the ARIMA(1,0) PACF, it seems not to peak over enough to claim to be an ARIMA(2,0) model.

Question 3

```
# get the data
library(astsa)
```

```
##
## Attaching package: 'astsa'

## The following object is masked from 'package:forecast':
##
##     gas
```

```
# make models
ar.ols(cmort,order= 2) -> cmort.ar2
arima(cmort, order = c(2,0,0)) -> cmort.arima200
cmort.ar2$x.intercept
```

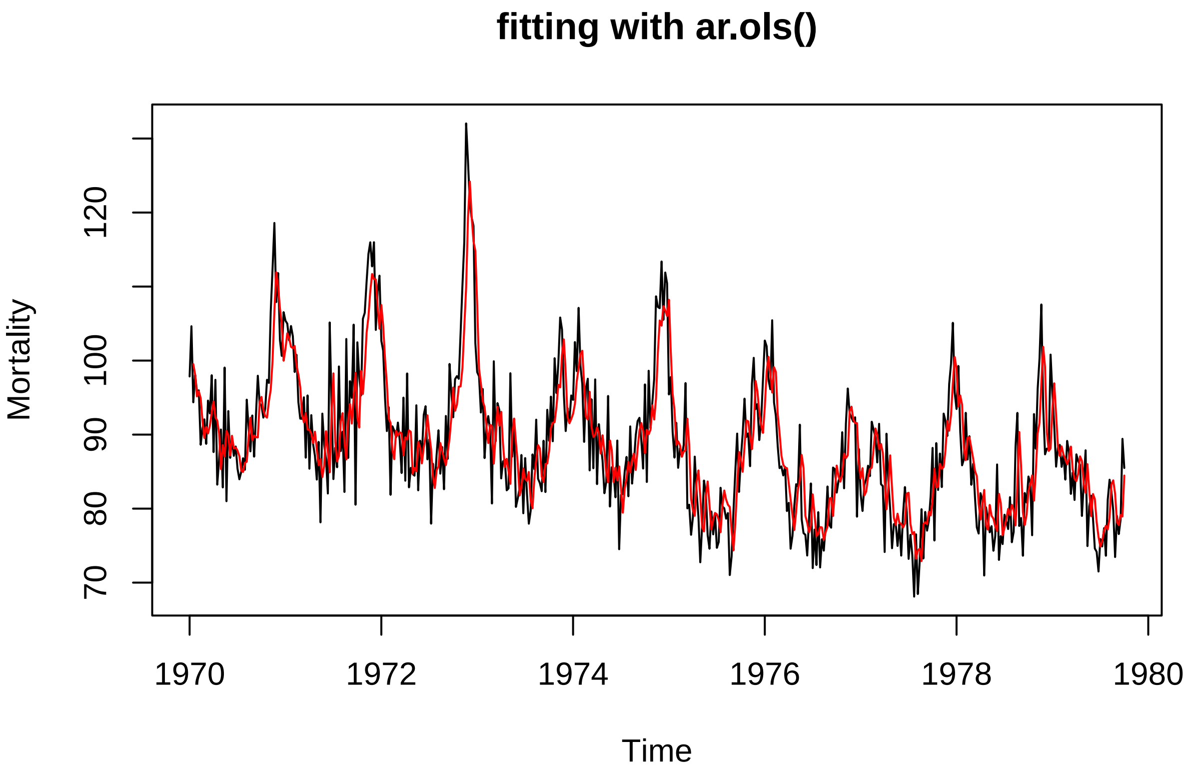
```
## [1] -0.04671956
```

```
summary(cmort.arima200)
```

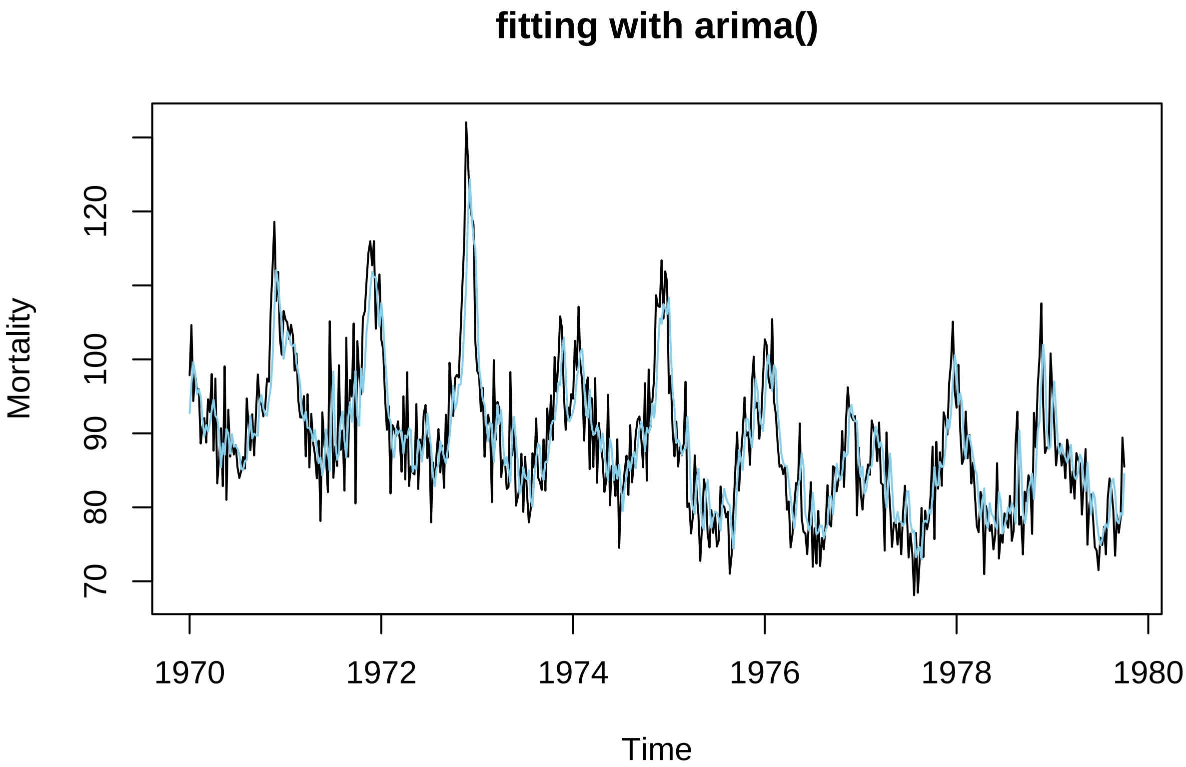
```
##
## Call:
## arima(x = cmort, order = c(2, 0, 0))
##
## Coefficients:
##          ar1     ar2  intercept
```

```
##        0.4301  0.4424    88.8538
## s.e.  0.0397  0.0398     1.9407
##
## sigma^2 estimated as 32.37:  log likelihood = -1604.71,  aic = 3217.43
##
## Training set error measures:
##                        ME     RMSE      MAE        MPE    MAPE     MASE
## Training set -0.04047717 5.689543 4.493995 -0.4445741 5.06128 0.846808
##                      ACF1
## Training set -0.01043152
```

```r
ts.plot(cmort, main = "fitting with ar.ols()", ylab = "Mortality")
lines(fitted(cmort.ar2),col = 'red')
```



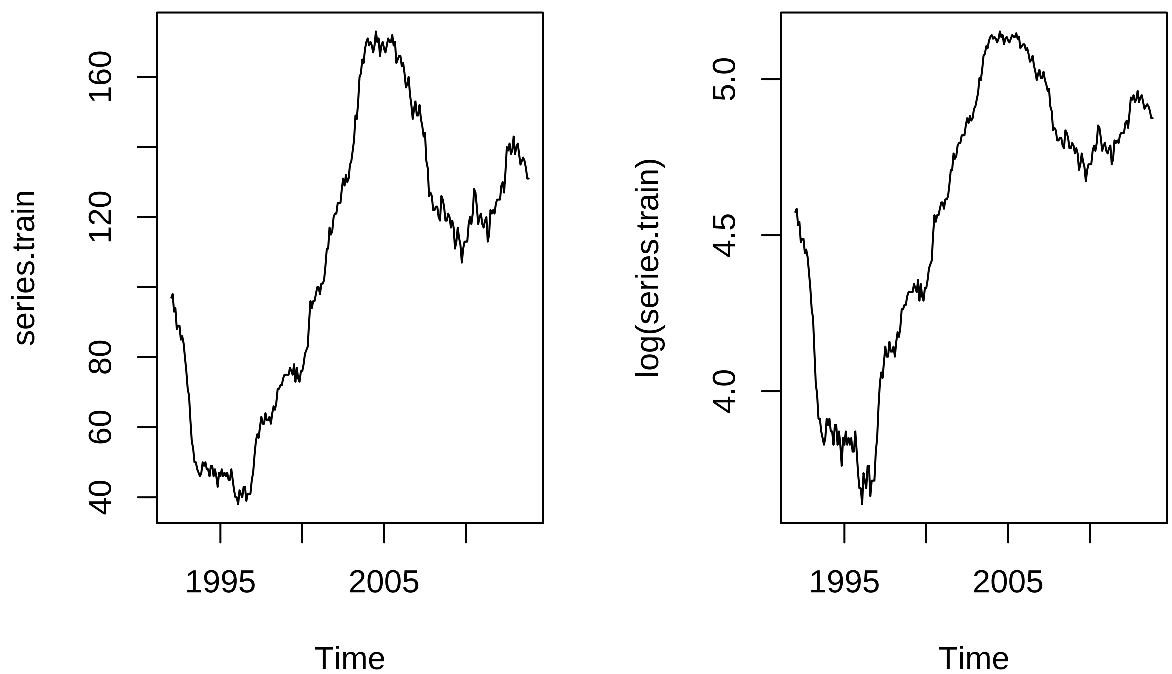

```r
ts.plot(cmort, main = "fitting with arima()", ylab = "Mortality")
lines(fitted(cmort.arima200),col = 'skyblue')
```



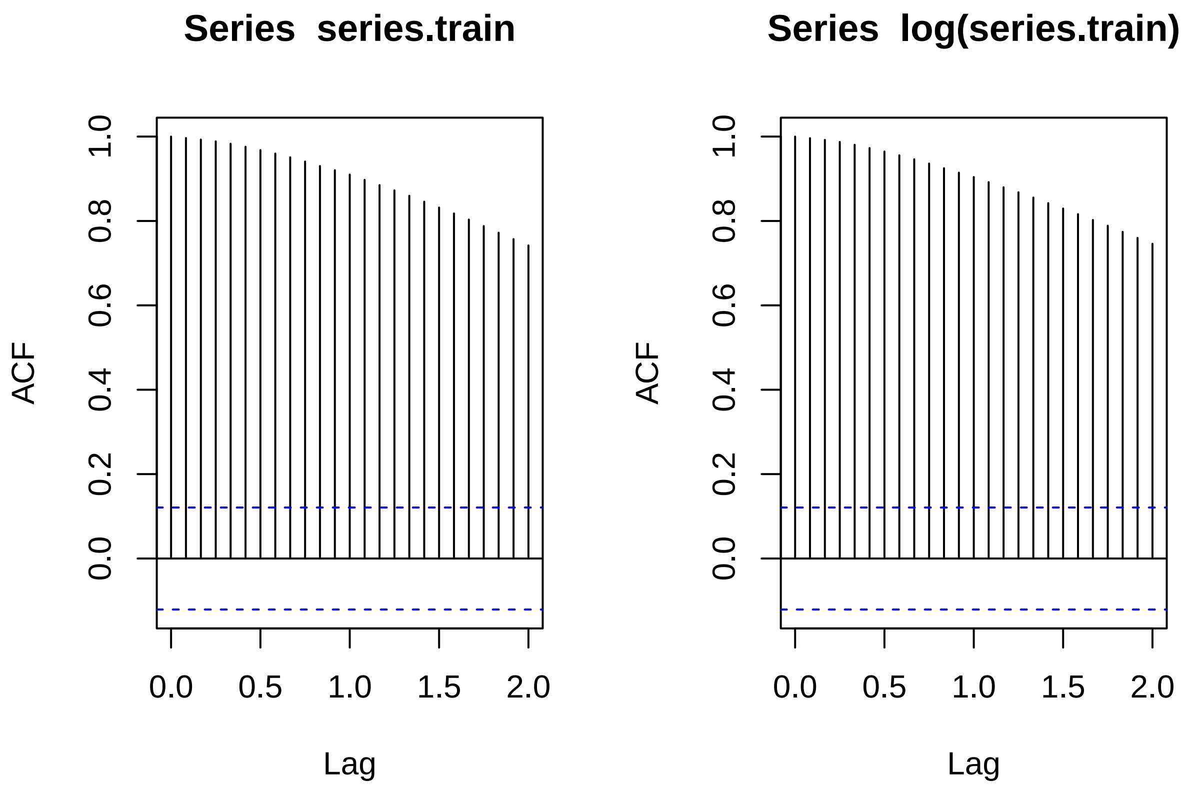
Question 4

part (a)

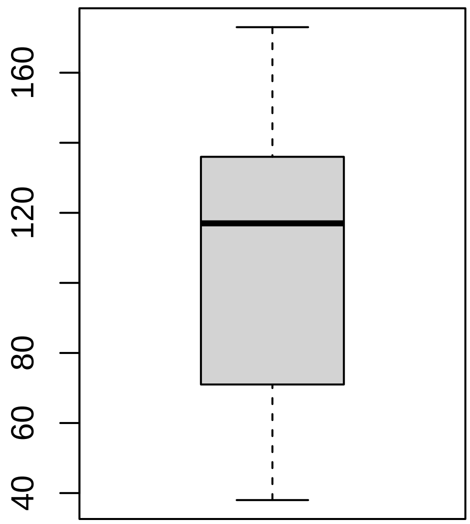
```
# getting data
demand <- as.vector(read.csv("Demand-2.txt", head = T)[,1])
series <- ts(demand, start =c(1992,1), frequency = 12)
series.train <- ts(demand[1:263], start =c(1992,1), frequency = 12)
series.val <- ts(demand[264:287], start =c(2014,1), frequency = 12)

par(mfrow=c(1,2))
plot.ts(series.train)
plot.ts(log(series.train))
```

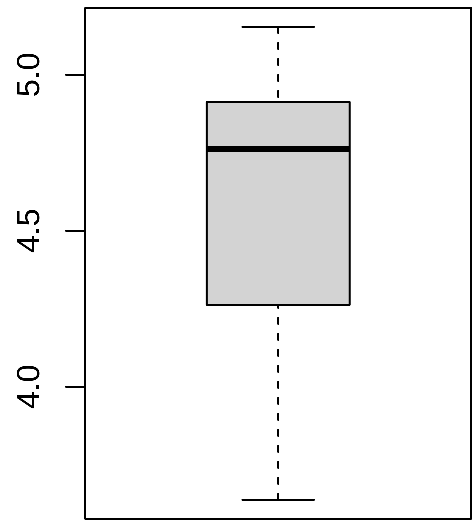



```r
acf(series.train)
acf(log(series.train))
```



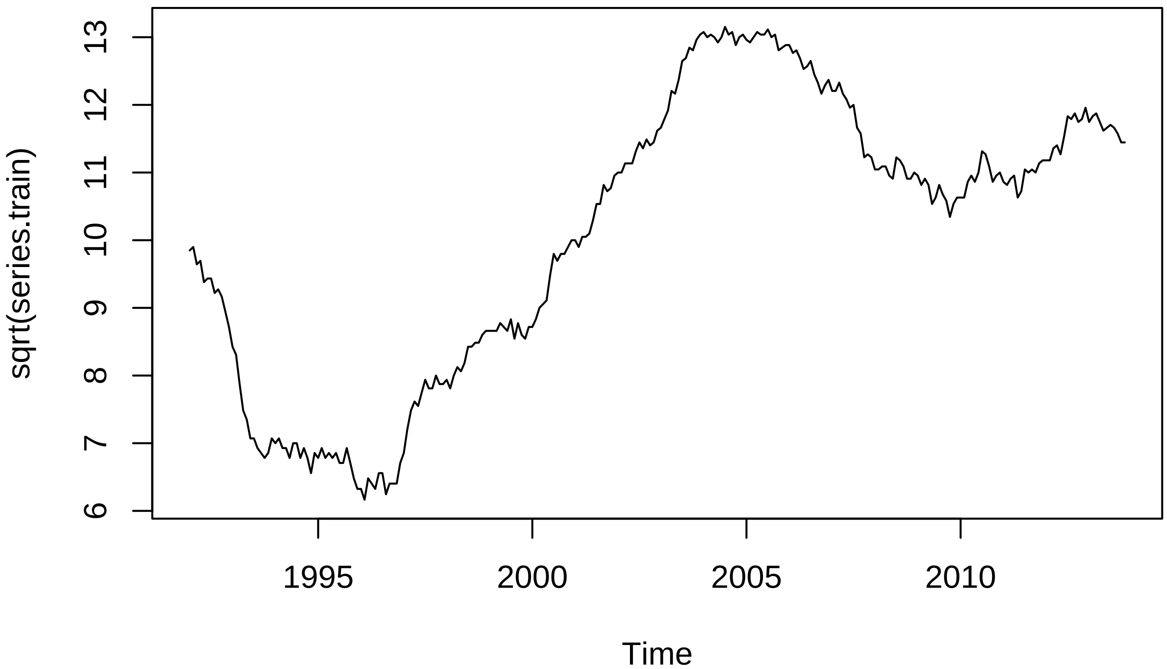

```r
boxplot(series.train)
boxplot(log(series.train))
```



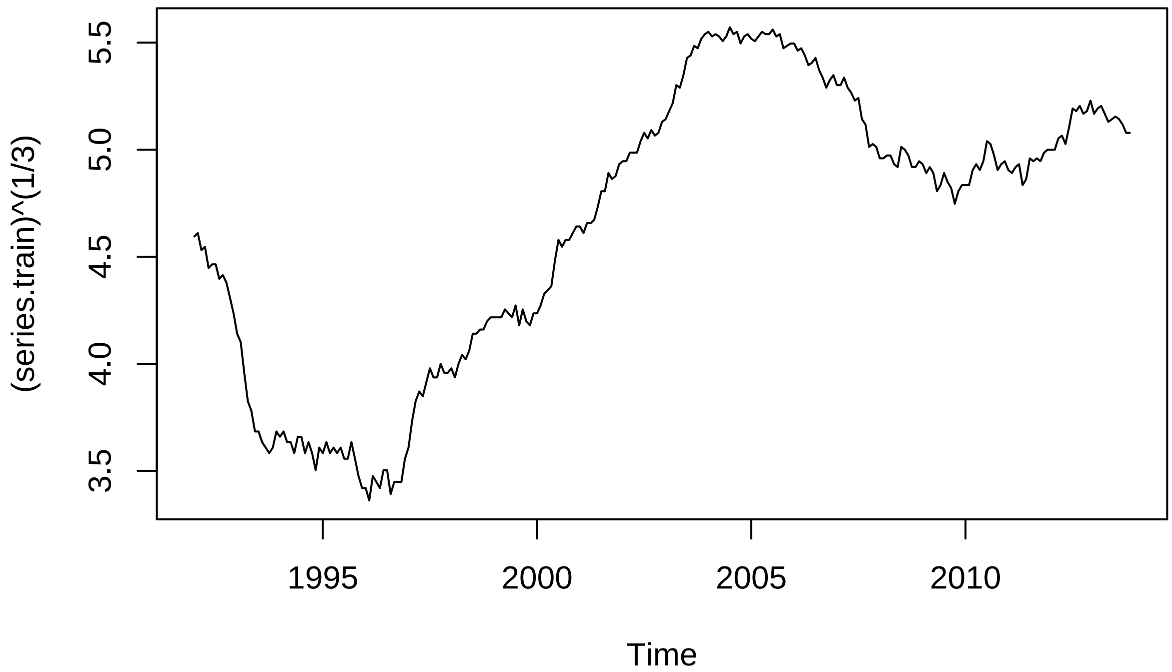


Going to try more transforms

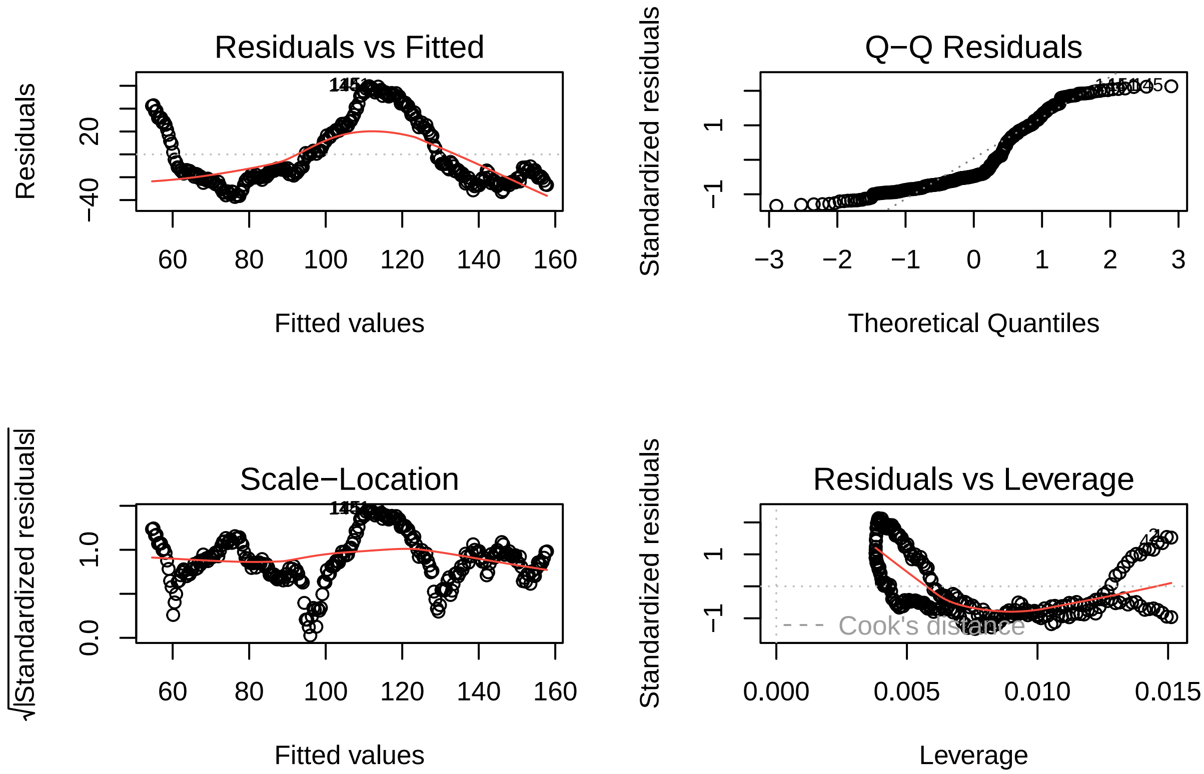
```
plot.ts(sqrt(series.train))
```



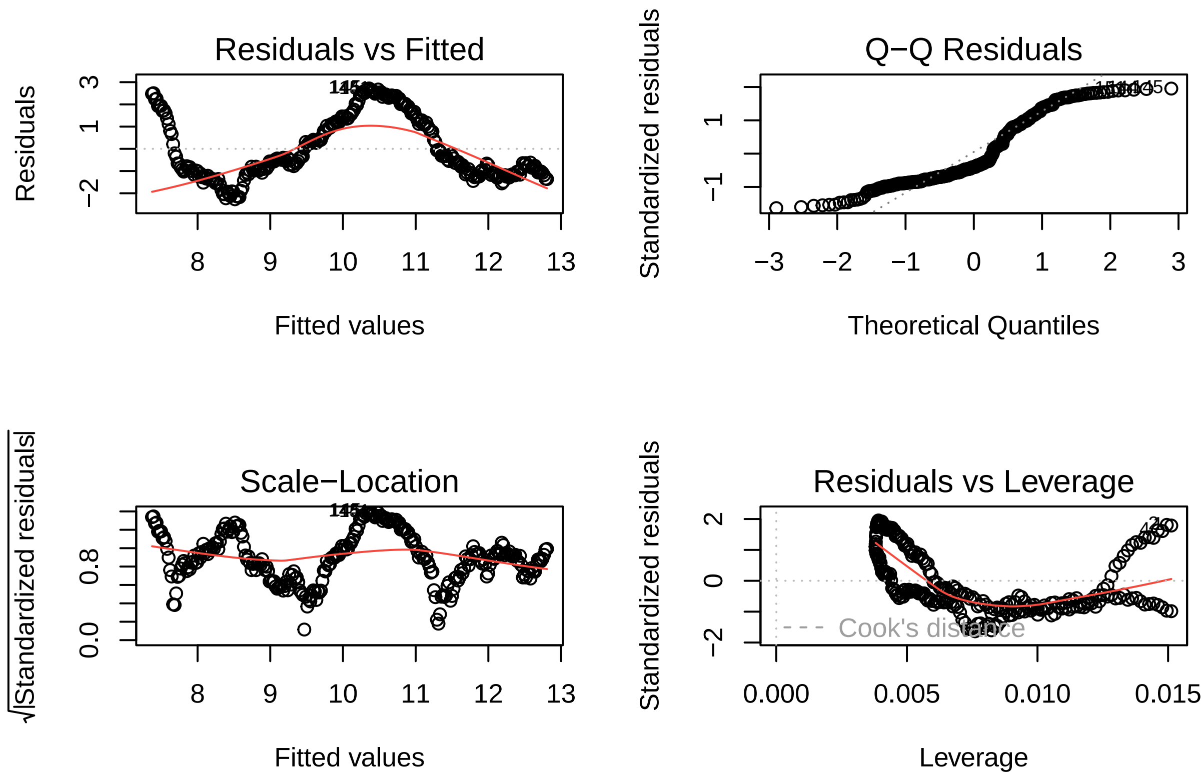

```
plot.ts((series.train)^(1/3))
```



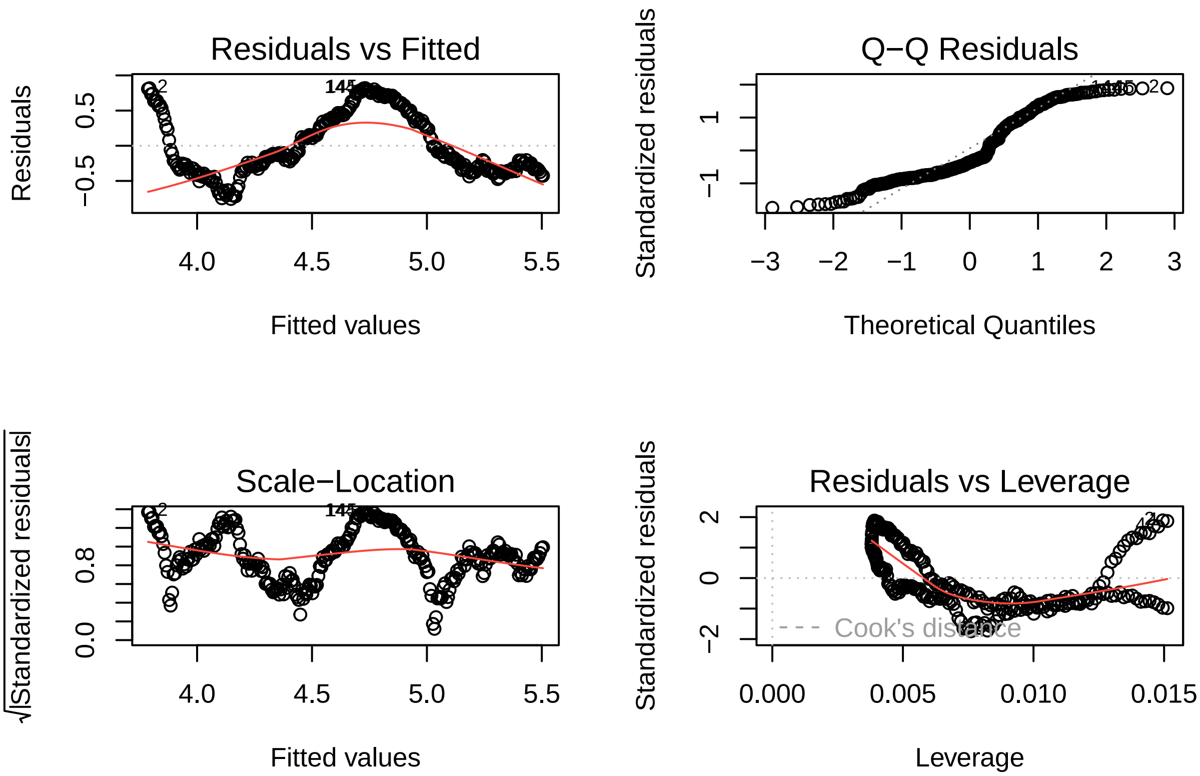

```r
# fit linear model :/
t = time(series.train)
fit0 <- lm(series.train ~ t)
fit1 <- lm(series.train ~ t + t^2)
fit2 <- lm(series.train ~ t + t^2 + t^3)
fit3 <- lm(series.train ~ t + t^2 + t^3 + t^4)
fit4 <- lm(sqrt(series.train) ~ t)
fit5 <- lm((series.train)^(1/3) ~ t)

par(mfrow = c(2,2))
plot(fit0)
plot(fit1)
```




```r
plot(fit2)
plot(fit3)
plot(fit4)
```



```r
plot(fit5)
```



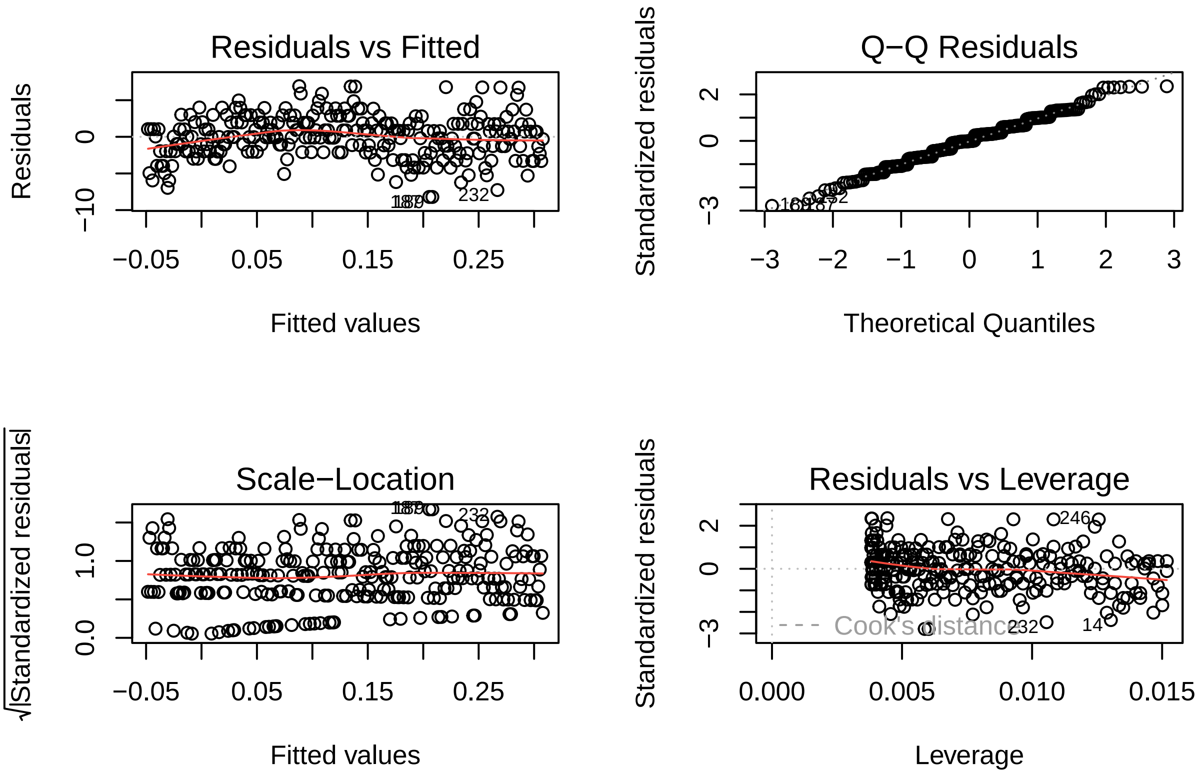
the scale-location line doesn't seem to straighten out for any linear fit we do. I am going to difference.

```r
#difference once

series.trainD <- diff(series.train,diff = 1)
# run kpss test
series.trainD_decomp <- decompose(series.trainD,type = c("additive"))
series.trainD_decomp <- na.omit(series.trainD_decomp)
kpss.test(series.trainD_decomp$random)
```

```
## Warning in kpss.test(series.trainD_decomp$random): p-value greater than printed
## p-value

##
##  KPSS Test for Level Stationarity
##
## data:  series.trainD_decomp$random
## KPSS Level = 0.0092476, Truncation lag parameter = 5, p-value = 0.1
```

The test concludes a failure to reject null, there is a chance that the system is stationary, so lets check variance again.

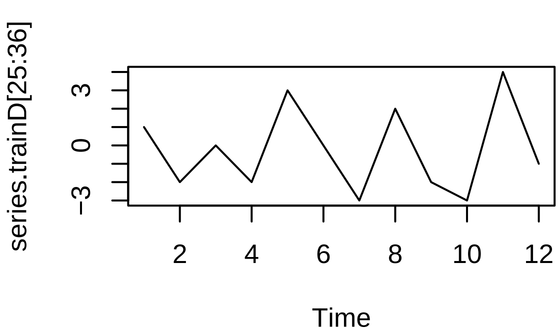
```r
par(mfrow=c(2,2))
plot(lm(series.trainD ~ time(series.trainD)))
```

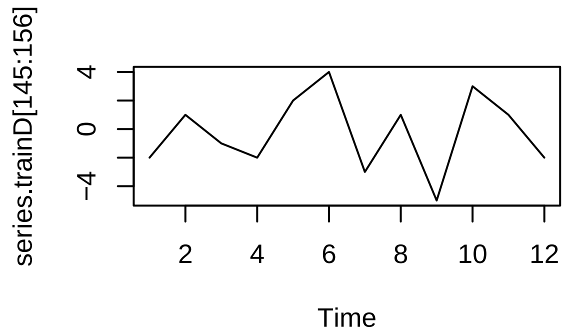


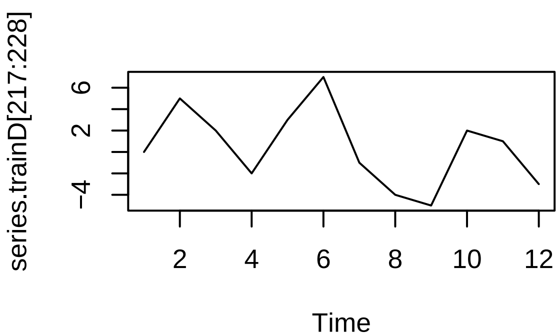
Yay. It is stabalized now!

part (b)

```
# render graphs the single out random cycles of 12
par(mfrow=c(2,2))
plot.ts(series.trainD[25:36])
plot.ts(series.trainD[145:156])
plot.ts(series.trainD[217:228])
```

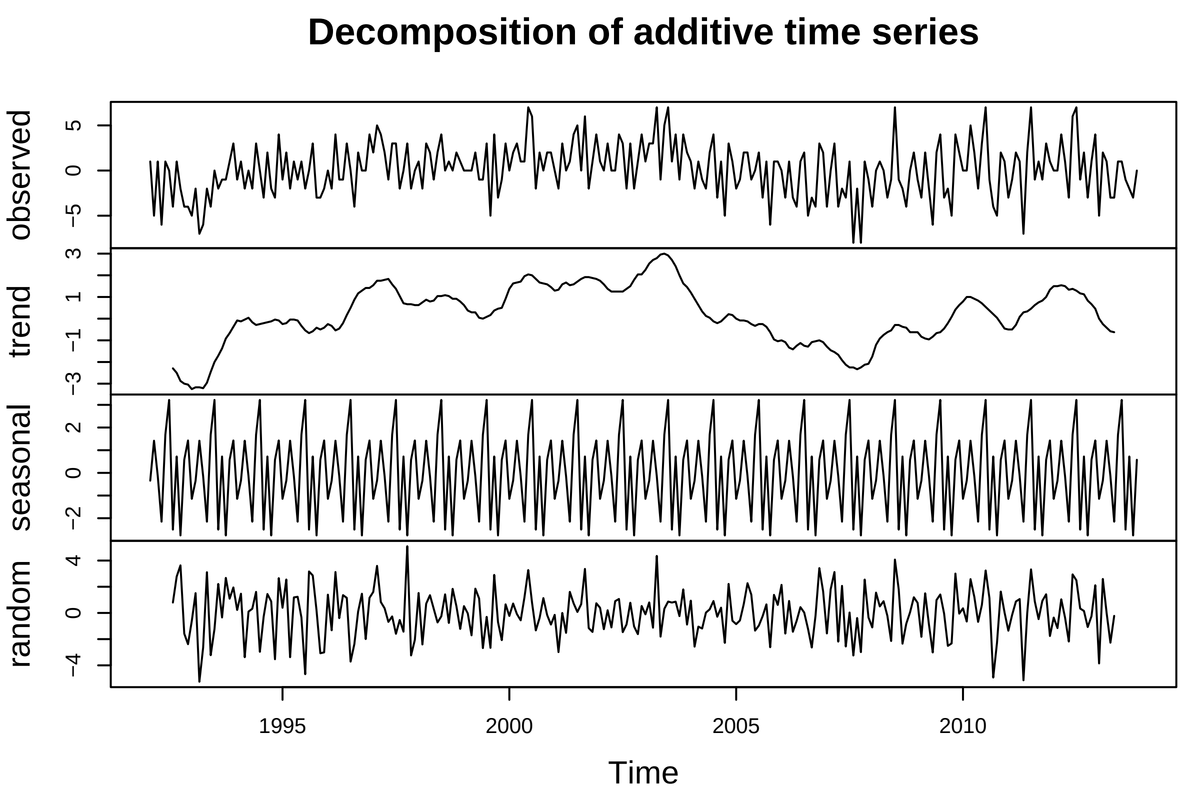






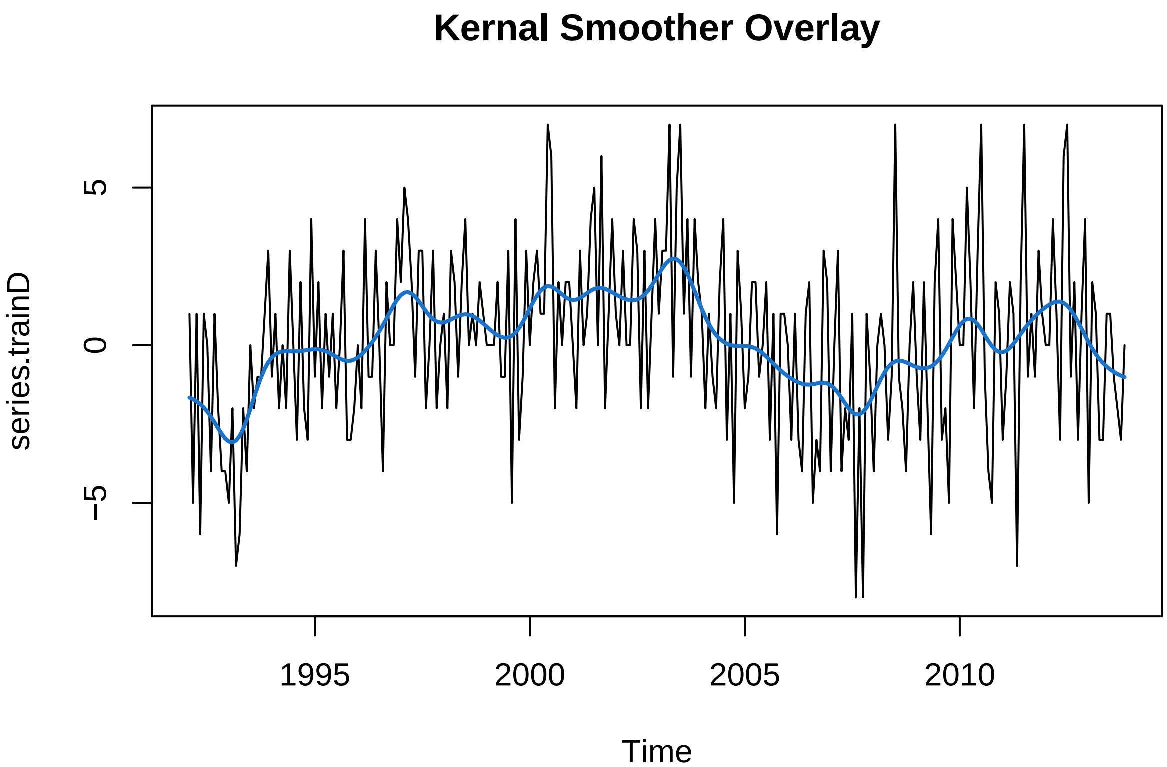
It seems in the later months, there are peaks at the 2nd, 6th, and 10th month while there are dips in the 4th, 7th, 9th and 12th month. This is not consistent in the earlier months.

```
# decomposing the series.train (we have done this previously in (a))
plot(series.trainD_decomp)
```



I feel it necessary to try a moving average smoother to find a trend less erratic.

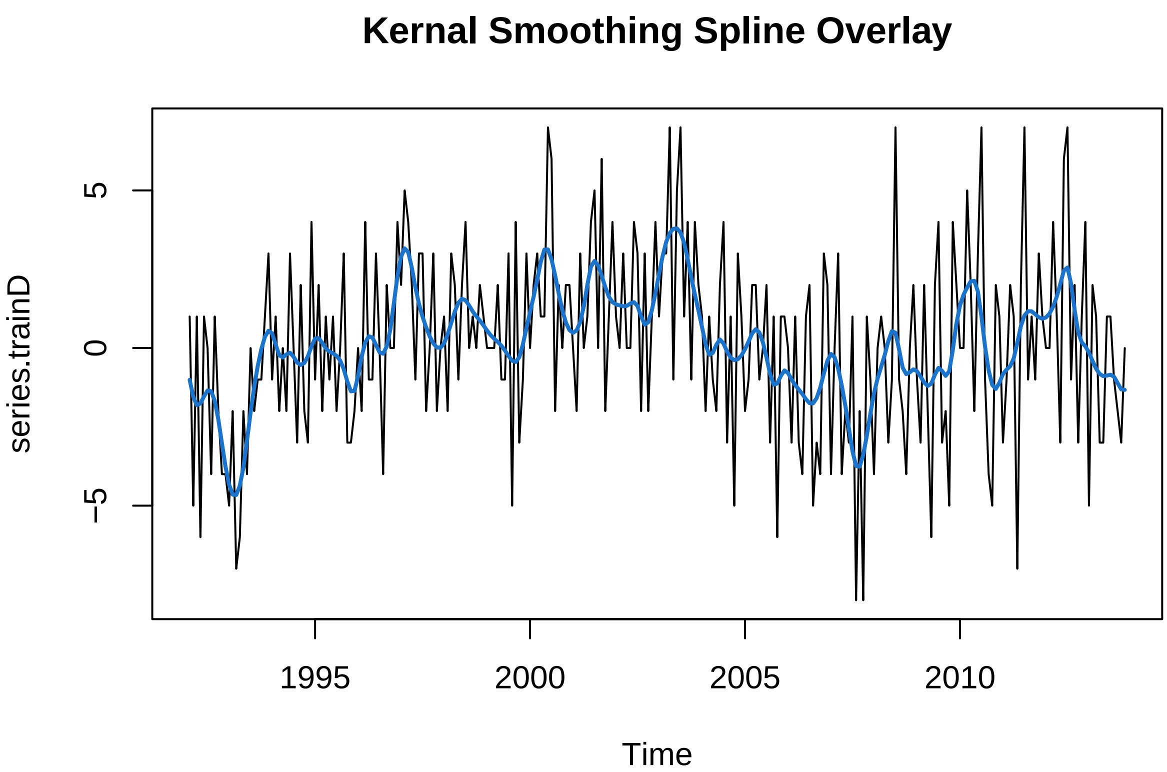
```
# trying kernal smoothing from the book pp.74
plot(series.trainD, main= "Kernal Smoother Overlay")
lines(ksmooth(time(series.trainD), series.trainD, "normal", bandwidth=1), lwd=2, col=4)
```




```
par(fig = c(.65, 1, .65, 1), new = TRUE) # the insert
gauss = function(x) { 1/sqrt(2*pi) * exp(-(x^2)/2) }
```

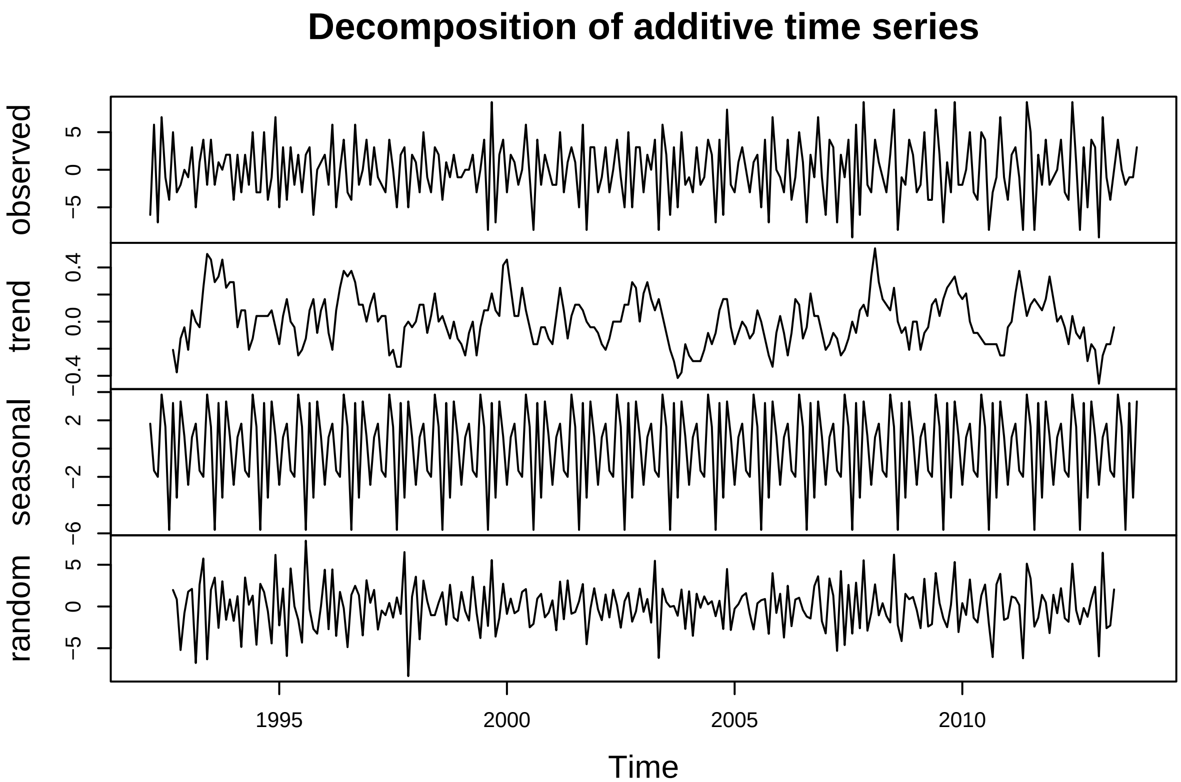
This looks close to what we got from the decomposed series before. Going to try a smoothing spline for this task.

```
plot(series.trainD, main= "Kernal Smoothing Spline Overlay")
lines(smooth.spline(time(series.trainD), series.trainD, spar=.3), lwd=2, col=4)
```



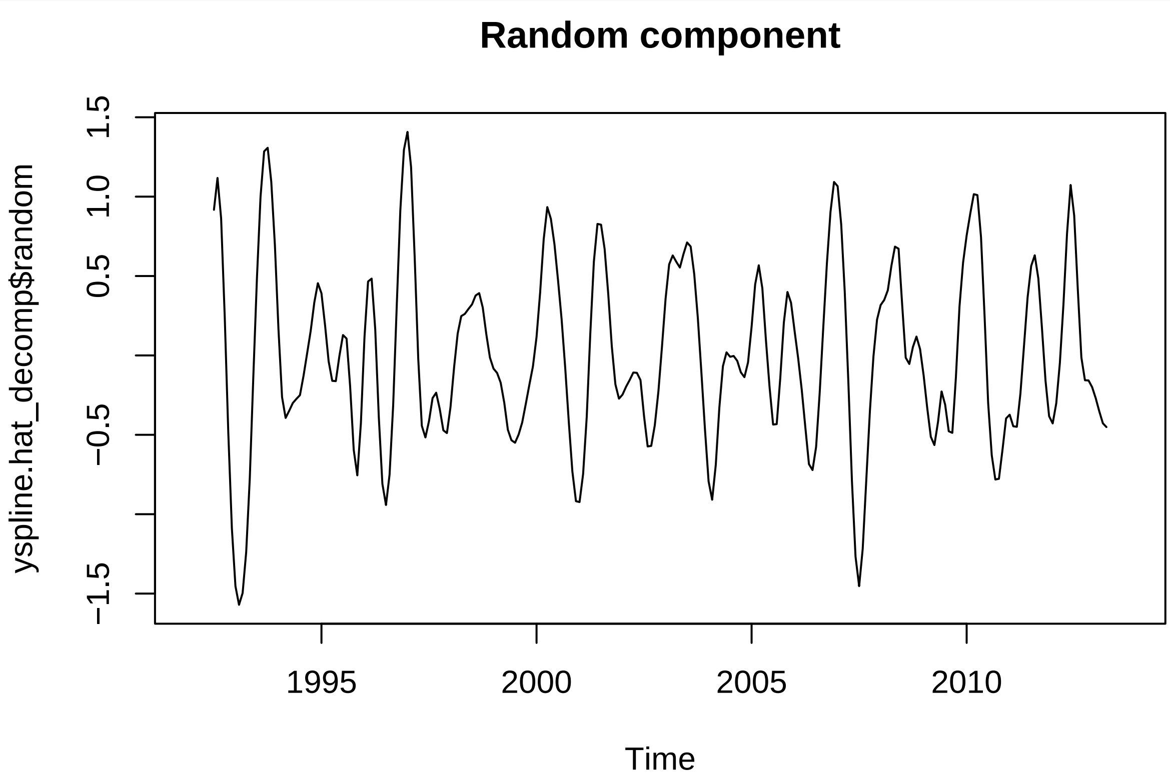
Let's see how this compares to differencing a decomposing one more time.

```
series.trainD2 <- diff(series.train,diff = 2)
series.trainD2_decomp <- decompose(series.trainD2,type = c("additive"))
series.trainD2_decomp <- na.omit(series.trainD2_decomp)
plot(series.trainD2_decomp)
```

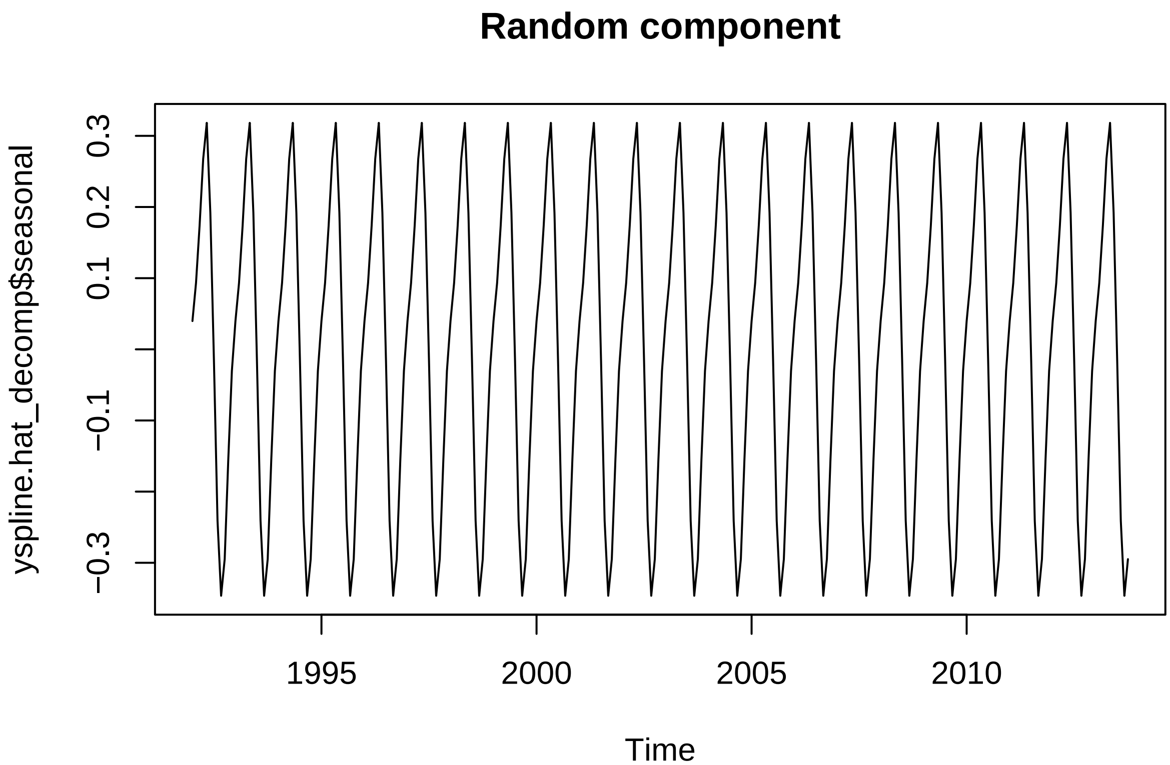


That made the general trend more spiked; let's stick with the MA kernal smoother.

```
yspline.hat <- ts(smooth.spline(time(series.trainD), series.trainD, spar=.3)$y, start =c(1992,1), frequ
yspline.hat_decomp <- decompose(yspline.hat,type = c("additive"))
yspline.hat_decomp <- na.omit(yspline.hat_decomp)
plot(yspline.hat_decomp$random, main = "Random component")
```



```r
plot(yspline.hat_decomp$seasonal, main = "Random component")
```




```r
# testing both for stationarity
# residuals
yspline.hat_decompR <- na.omit(yspline.hat_decomp$random)
kpss.test(yspline.hat_decompR)
```

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

```
##
##  KPSS Test for Level Stationarity
##
## data:  yspline.hat_decompR
## KPSS Level = 0.0096878, Truncation lag parameter = 5, p-value = 0.1
```

```r
# seasonal
yspline.hat_decompS <- na.omit(yspline.hat_decomp$seasonal)
kpss.test(yspline.hat_decompS)
```

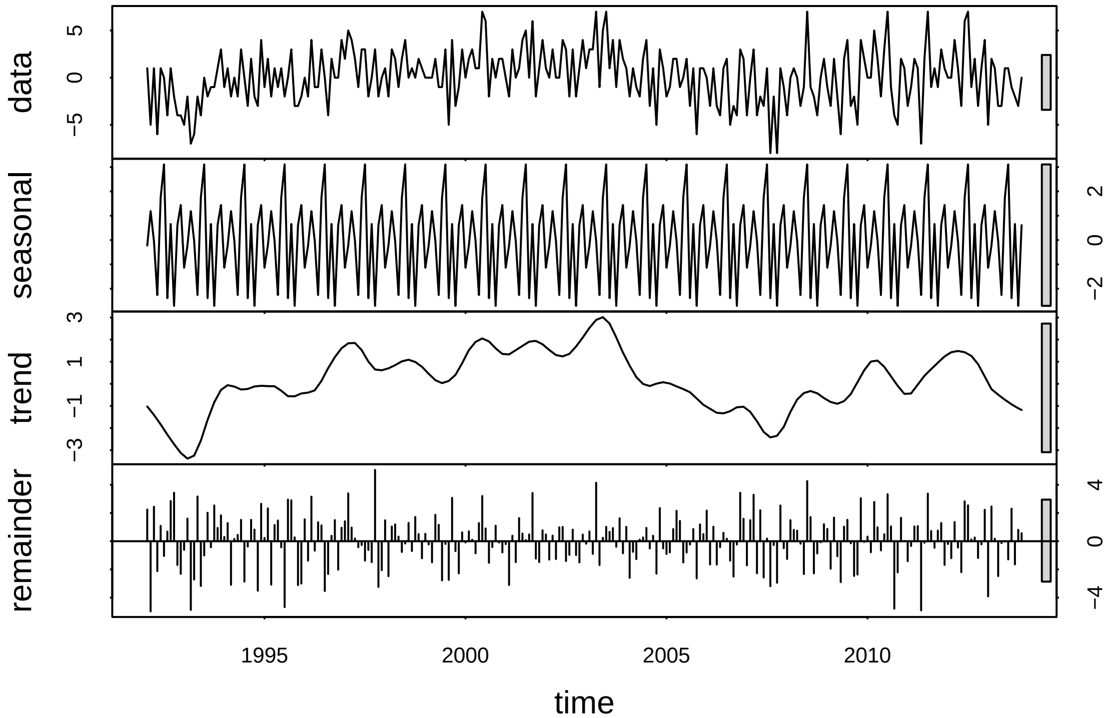
```
## Warning in kpss.test(yspline.hat_decompS): p-value greater than printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  yspline.hat_decompS
## KPSS Level = 0.011337, Truncation lag parameter = 5, p-value = 0.1
```

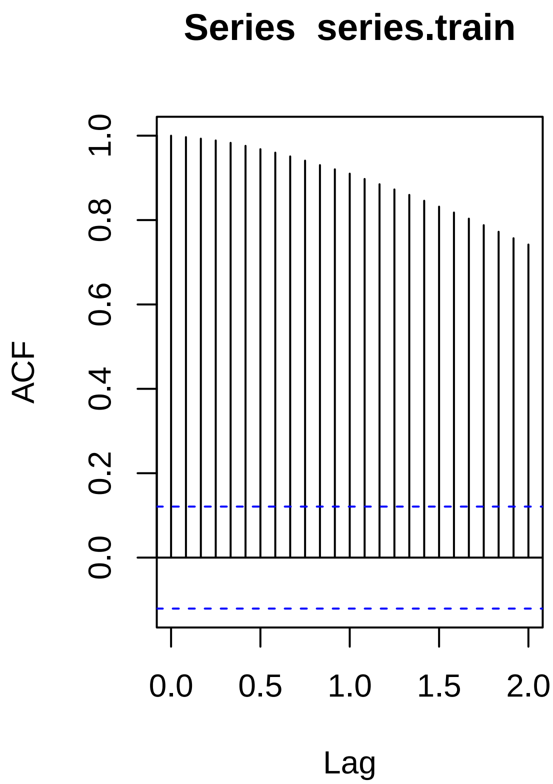
Both test statistics fall in the the "Fail To Reject" region; both systems are stationary.

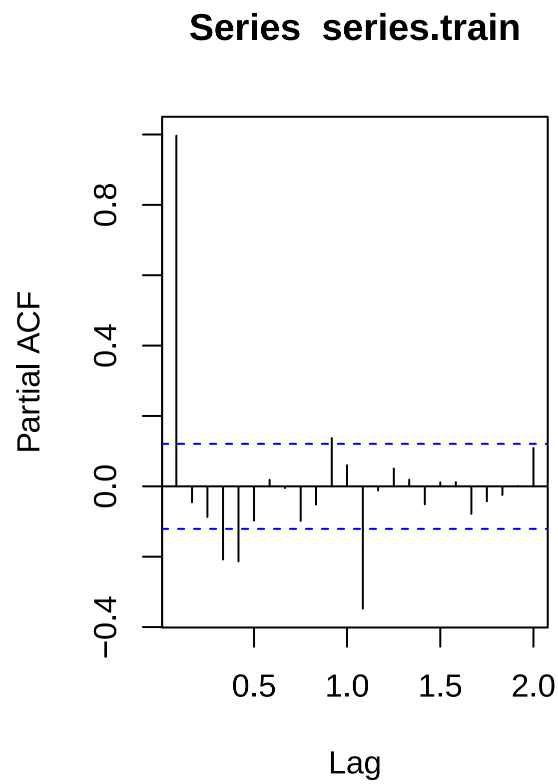
part (c)

```
# let's deseason
series.trainD.Decomp <- stl(series.trainD, s.window = "periodic")
plot(series.trainD.Decomp)
```

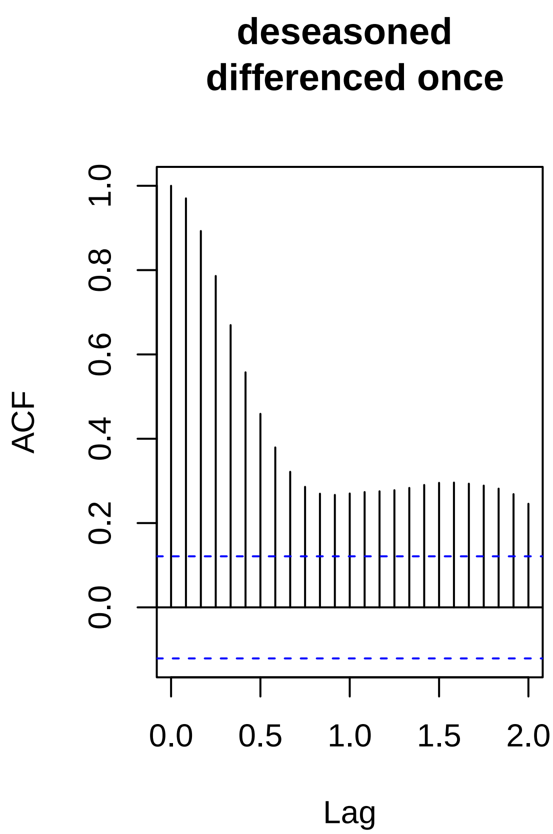


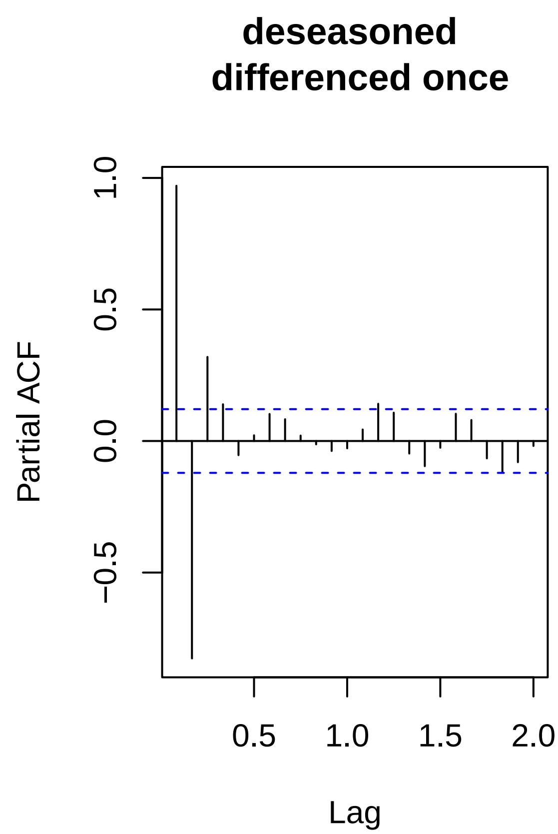

```
par(mfrow = c(1,2))
acf(series.train)
pacf(series.train)
```



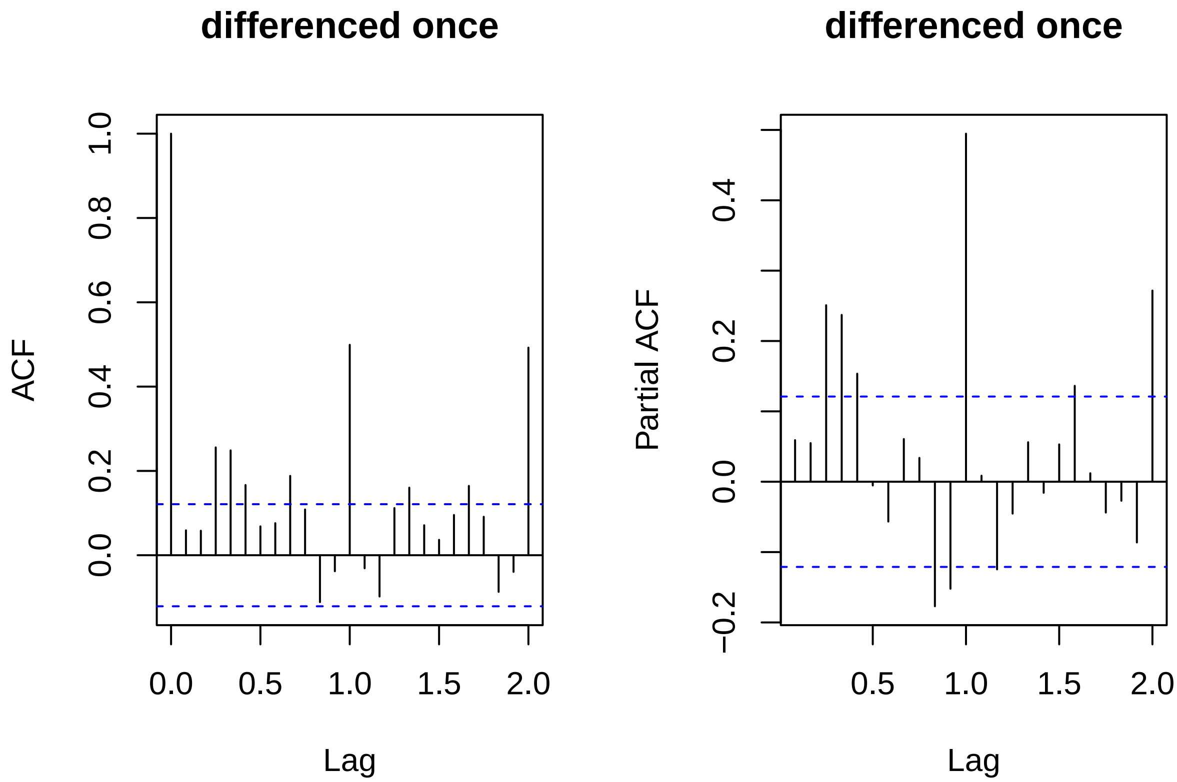



```r
acf(yspline.hat, main = "deseasoned \n differenced once")
pacf(yspline.hat, main = "deseasoned \n differenced once")
```





```r
acf(series.trainD, main = "differenced once")
pacf(series.trainD, main = "differenced once")
```



I believe it fair to suggest that $1 \leq p \leq 3$ and $q = 0$ from the deseasoned graphs. Looking at the seasoned graphs, PACF seems to suggest that q could be 1.

We saw previously, after differencing once, seasonality every 12 months ($s = 12$); let's make $d = 1$.

We notice a spike in both graphs at every whole number (note the view of the graph cuts of after 2.0). This can imply $P \geq 2$ and $Q \geq 2$. I did not conduct seasonal differencing thus $D = 0$.

part (d)

The view of the ACF and PACF graphs cuts off after 2.0. There could be more spikes at later lags.

Let's use auto.arima() to comb through these parameters.

$p \in [1, 3]$ $q = 1$ $d = 1$ $P = \in [2, 4]$ $Q = \in [2, 4]$ $D = 0$

Note P,Q will only go up to four because I want to save RAM.

```r
# running auto.arima
auto.arima(
  series.train,
  d = 1,
  D = 0,
  max.p = 3,
  max.q = 1,
  max.P = 4,
  max.Q = 4,
  max.d = 1,
  max.D = 0,
```

```r
  start.p = 1,
  start.q = 1,
  start.P = 2,
  start.Q = 2,
  stepwise = T)
```

```
## Series: series.train
## ARIMA(1,1,0)(3,0,1)[12]
##
## Coefficients:
##          ar1     sar1    sar2    sar3    sma1
##       0.1933  -0.1511  0.4526  0.3702  0.4180
## s.e.  0.2339   0.0344  0.0393  0.0707  0.1166
##
## sigma^2 = 5.007:  log likelihood = -585.45
## AIC=1182.91   AICc=1183.24   BIC=1204.32
```

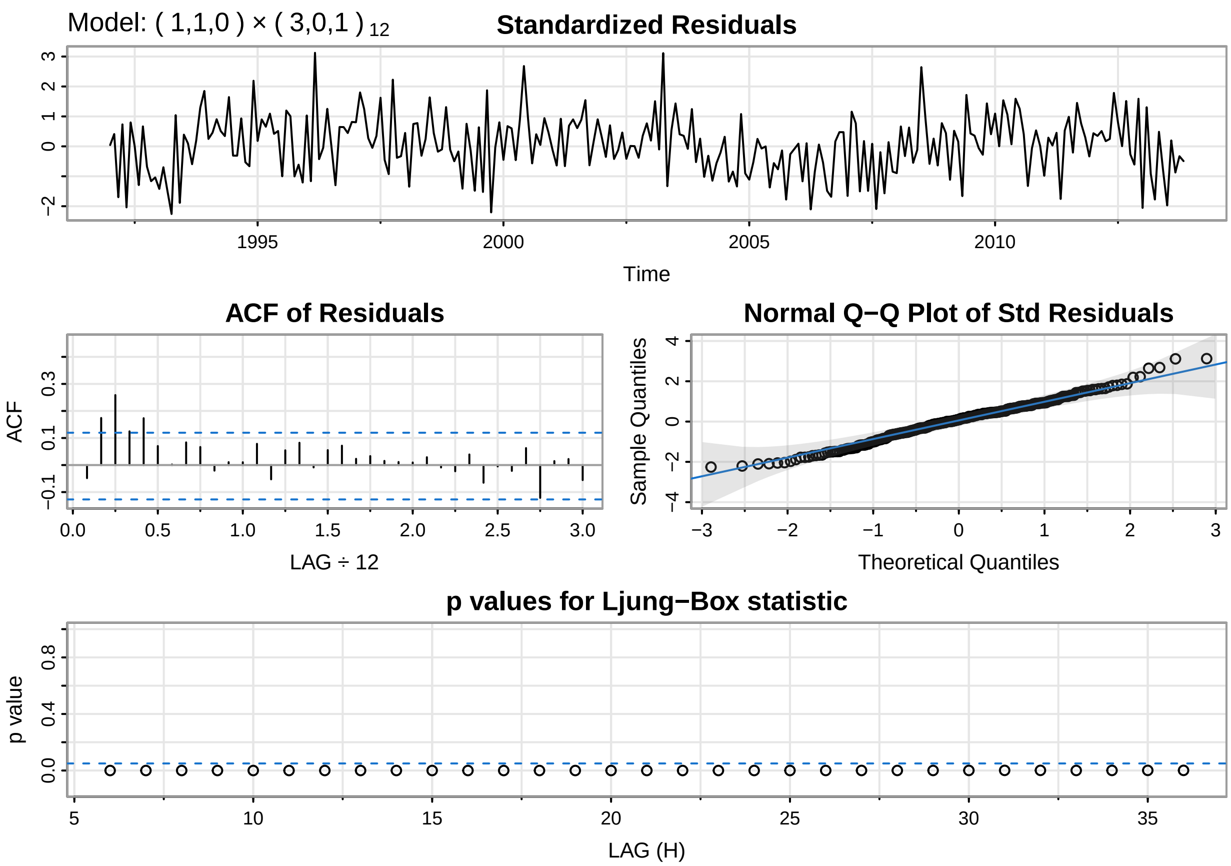
We have an AIC of 1182.91 auto.arima is telling us we should look into (1,1,0)(3,0,1)[12].

It is strange that this auto.arima() function is suggesting that q = 0 and Q = 1 but those parameters are not within the ranges I input.

```r
# residual diagnostics
sys.auto <- arima(series.train,order = c(1,1,0),seasonal = list(order = c(3,0,1),period = 12))
sarima(series.train, p = 1, q = 0, d =1, P = 3, D = 0, Q = 1, S = 12)
```

```
## initial  value 1.070751
## iter   2 value 0.882455
## iter   3 value 0.818058
## iter   4 value 0.792314
## iter   5 value 0.779792
## iter   6 value 0.779023
## iter   7 value 0.778532
## iter   8 value 0.776314
## iter   9 value 0.774390
## iter  10 value 0.772440
## iter  11 value 0.771119
## iter  12 value 0.770474
## iter  13 value 0.770425
## iter  14 value 0.770417
## iter  15 value 0.770414
## iter  16 value 0.770405
## iter  17 value 0.770381
## iter  18 value 0.770375
## iter  19 value 0.770373
## iter  20 value 0.770370
## iter  21 value 0.770367
## iter  22 value 0.770366
## iter  23 value 0.770366
## iter  23 value 0.770366
## iter  23 value 0.770366
## final  value 0.770366
## converged
## initial  value 0.821775
## iter   2 value 0.821157
## iter   3 value 0.819686
```

```
## iter   4 value 0.819293
## iter   5 value 0.818039
## iter   6 value 0.817087
## iter   7 value 0.816682
## iter   8 value 0.816560
## iter   9 value 0.815617
## iter  10 value 0.815516
## iter  11 value 0.815457
## iter  12 value 0.815320
## iter  13 value 0.815315
## iter  14 value 0.815311
## iter  15 value 0.815301
## iter  16 value 0.815298
## iter  17 value 0.815295
## iter  17 value 0.815303
## iter  17 value 0.815301
## final  value 0.815295
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE t.value p.value
## ar1        0.1946 0.0176 11.0480  0.0000
## sar1      -0.1715 0.0275 -6.2440  0.0000
## sar2       0.4604 0.0416 11.0693  0.0000
## sar3       0.3783 0.0128 29.5860  0.0000
## sma1       0.4348 0.0626  6.9484  0.0000
## constant  -0.2332 0.5915 -0.3942  0.6938
##
## sigma^2 estimated as 4.906193 on 256 degrees of freedom
##
## AIC = 4.521902  AICc = 4.523159  BIC = 4.617239
##
```




```
Box.test(residuals(sys.auto), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys.auto)
## X-squared = 52.494, df = 20, p-value = 9.643e-05
```

```
sprintf("this gives and AIC of %f", sys.auto$aic)
```

```
## [1] "this gives and AIC of 1182.908227"
```

Auto.arima() suggested a model that does not do well according to the Box-Ljung test.

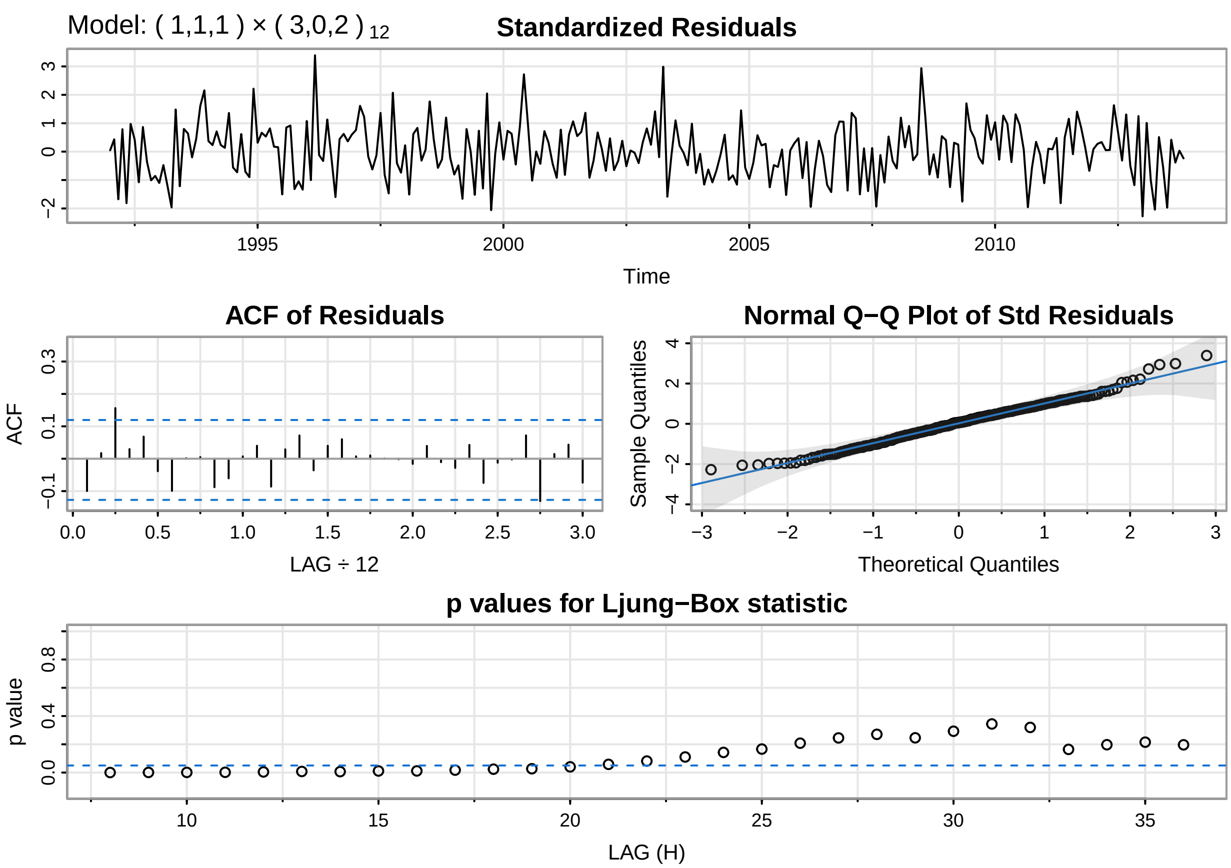
Using intuition, make q = 1 and Q = 2 and test that.

```r
sys0 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(3,0,2),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 3, D = 0, Q = 2, S = 12)
```

```
## initial  value 1.070751
## iter   2 value 0.854049
## iter   3 value 0.853596
## iter   4 value 0.779203
## iter   5 value 0.777801
## iter   6 value 0.776526
## iter   7 value 0.773065
## iter   8 value 0.764280
## iter   9 value 0.759854
## iter  10 value 0.754085
## iter  11 value 0.743993
## iter  12 value 0.736760
## iter  13 value 0.735292
## iter  14 value 0.731532
## iter  15 value 0.730598
## iter  16 value 0.729373
## iter  17 value 0.727659
## iter  18 value 0.726487
## iter  19 value 0.726312
## iter  20 value 0.726217
## iter  21 value 0.726121
## iter  22 value 0.726046
## iter  23 value 0.725977
## iter  24 value 0.725948
## iter  25 value 0.725938
## iter  26 value 0.725924
## iter  27 value 0.725901
## iter  28 value 0.725880
## iter  29 value 0.725870
## iter  30 value 0.725867
## iter  31 value 0.725865
## iter  32 value 0.725861
## iter  33 value 0.725853
## iter  34 value 0.725835
## iter  35 value 0.725805
## iter  36 value 0.725794
## iter  37 value 0.725787
## iter  38 value 0.725785
## iter  39 value 0.725778
## iter  40 value 0.725777
## iter  41 value 0.725777
## iter  42 value 0.725777
## iter  43 value 0.725775
## iter  44 value 0.725772
## iter  45 value 0.725765
## iter  46 value 0.725758
## iter  47 value 0.725754
## iter  48 value 0.725753
## iter  49 value 0.725753
## iter  50 value 0.725753
## iter  51 value 0.725752
```

```
## iter  52 value 0.725752
## iter  53 value 0.725752
## iter  54 value 0.725752
## iter  55 value 0.725751
## iter  56 value 0.725751
## iter  57 value 0.725751
## iter  58 value 0.725751
## iter  58 value 0.725751
## iter  58 value 0.725751
## final  value 0.725751
## converged
## initial  value 0.764579
## iter   2 value 0.764116
## iter   3 value 0.762142
## iter   4 value 0.761992
## iter   5 value 0.761956
## iter   6 value 0.761931
## iter   7 value 0.761895
## iter   8 value 0.761800
## iter   9 value 0.761743
## iter  10 value 0.761723
## iter  11 value 0.761711
## iter  12 value 0.761683
## iter  13 value 0.761636
## iter  14 value 0.761563
## iter  15 value 0.761498
## iter  16 value 0.761465
## iter  17 value 0.761461
## iter  18 value 0.761458
## iter  19 value 0.761458
## iter  20 value 0.761456
## iter  21 value 0.761452
## iter  22 value 0.761449
## iter  23 value 0.761447
## iter  24 value 0.761447
## iter  25 value 0.761447
## iter  26 value 0.761446
## iter  27 value 0.761446
## iter  28 value 0.761445
## iter  29 value 0.761445
## iter  30 value 0.761445
## iter  31 value 0.761445
## iter  32 value 0.761444
## iter  33 value 0.761444
## iter  34 value 0.761444
## iter  35 value 0.761444
## iter  36 value 0.761444
## iter  37 value 0.761444
## iter  37 value 0.761444
## iter  37 value 0.761444
## final  value 0.761444
## converged
## <><><><><><><><><><><><><><>
##
```

```
## Coefficients:
##          Estimate     SE  t.value p.value
## ar1        0.9090 0.0465  19.5664  0.0000
## ma1       -0.7426 0.0709 -10.4700  0.0000
## sar1      -0.1018 0.2024  -0.5029  0.6155
## sar2       0.4206 0.2055   2.0469  0.0417
## sar3       0.3941 0.1518   2.5953  0.0100
## sma1       0.3182 0.2359   1.3485  0.1787
## sma2       0.0192 0.3035   0.0631  0.9497
## constant  -0.4147 1.3178  -0.3147  0.7532
##
## sigma^2 estimated as 4.376583 on 254 degrees of freedom
##
## AIC = 4.429467  AICc = 4.431639  BIC = 4.552044
##
```




```r
Box.test(residuals(sys0), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys0)
## X-squared = 22.974, df = 20, p-value = 0.29
```

```r
sprintf("this gives and AIC of %f", sys0$aic)
```

```
## [1] "this gives and AIC of 1158.622888"
```

This is a better AIC and Ljung-Box test gives a much more hopeful conclusion.

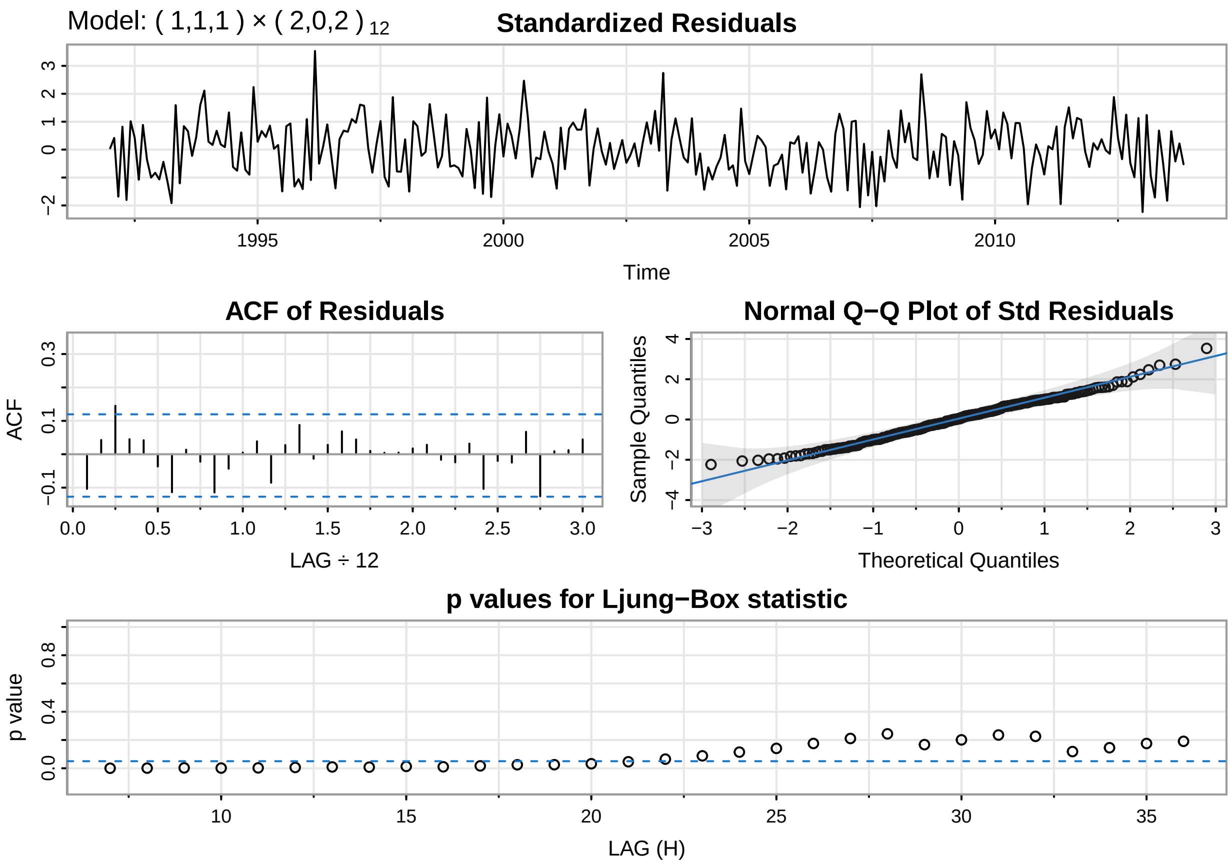
I am going to try different values for P and Q and see if I can get better results

```r
sys1 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(2,0,2),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 2, D = 0, Q = 2, S = 12)
```

```
## initial  value 1.061388
## iter   2 value 0.881128
## iter   3 value 0.827535
## iter   4 value 0.806374
## iter   5 value 0.801764
## iter   6 value 0.796577
## iter   7 value 0.792163
## iter   8 value 0.790923
## iter   9 value 0.789371
## iter  10 value 0.783957
## iter  11 value 0.781926
## iter  12 value 0.777939
## iter  13 value 0.762725
## iter  14 value 0.758488
## iter  15 value 0.751480
## iter  16 value 0.745737
## iter  17 value 0.744957
## iter  18 value 0.744206
## iter  19 value 0.743582
## iter  20 value 0.743490
## iter  21 value 0.743469
## iter  22 value 0.743450
## iter  23 value 0.743438
## iter  24 value 0.743433
## iter  25 value 0.743433
## iter  26 value 0.743433
## iter  27 value 0.743432
## iter  28 value 0.743432
## iter  29 value 0.743430
## iter  30 value 0.743429
## iter  31 value 0.743425
## iter  32 value 0.743423
## iter  33 value 0.743420
## iter  34 value 0.743417
## iter  35 value 0.743414
## iter  36 value 0.743412
## iter  37 value 0.743411
## iter  38 value 0.743411
## iter  38 value 0.743411
## iter  38 value 0.743411
## final  value 0.743411
## converged
## initial  value 0.773922
## iter   2 value 0.772814
## iter   3 value 0.769767
## iter   4 value 0.769653
## iter   5 value 0.769494
## iter   6 value 0.769183
## iter   7 value 0.768672
## iter   8 value 0.768452
## iter   9 value 0.768365
```

```
## iter  10 value 0.768285
## iter  11 value 0.768135
## iter  12 value 0.768006
## iter  13 value 0.767946
## iter  14 value 0.767938
## iter  15 value 0.767934
## iter  16 value 0.767922
## iter  17 value 0.767917
## iter  18 value 0.767913
## iter  19 value 0.767909
## iter  20 value 0.767894
## iter  21 value 0.767869
## iter  22 value 0.767835
## iter  23 value 0.767806
## iter  24 value 0.767790
## iter  25 value 0.767785
## iter  26 value 0.767778
## iter  27 value 0.767765
## iter  28 value 0.767755
## iter  29 value 0.767752
## iter  30 value 0.767751
## iter  31 value 0.767750
## iter  32 value 0.767750
## iter  33 value 0.767749
## iter  34 value 0.767748
## iter  35 value 0.767746
## iter  36 value 0.767745
## iter  37 value 0.767743
## iter  38 value 0.767743
## iter  39 value 0.767742
## iter  40 value 0.767742
## iter  41 value 0.767741
## iter  42 value 0.767741
## iter  43 value 0.767740
## iter  44 value 0.767740
## iter  45 value 0.767740
## iter  46 value 0.767740
## iter  47 value 0.767740
## iter  48 value 0.767740
## iter  48 value 0.767740
## iter  48 value 0.767740
## final  value 0.767740
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE  t.value p.value
## ar1        0.9070 0.0435  20.8352  0.0000
## ma1       -0.7161 0.0690 -10.3773  0.0000
## sar1       0.7357 0.2564   2.8691  0.0045
## sar2       0.1842 0.2426   0.7593  0.4484
## sma1      -0.5492 0.2563  -2.1425  0.0331
## sma2       0.0236 0.1795   0.1313  0.8956
## constant  -0.4072 1.5832  -0.2572  0.7972
```

```
##
## sigma^2 estimated as 4.438563 on 255 degrees of freedom
##
## AIC = 4.434426  AICc = 4.436109  BIC = 4.543384
##
```




```r
Box.test(residuals(sys1), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys1)
## X-squared = 25.085, df = 20, p-value = 0.1982
```

```r
sprintf("this gives and AIC of %f", sys1$aic)
```

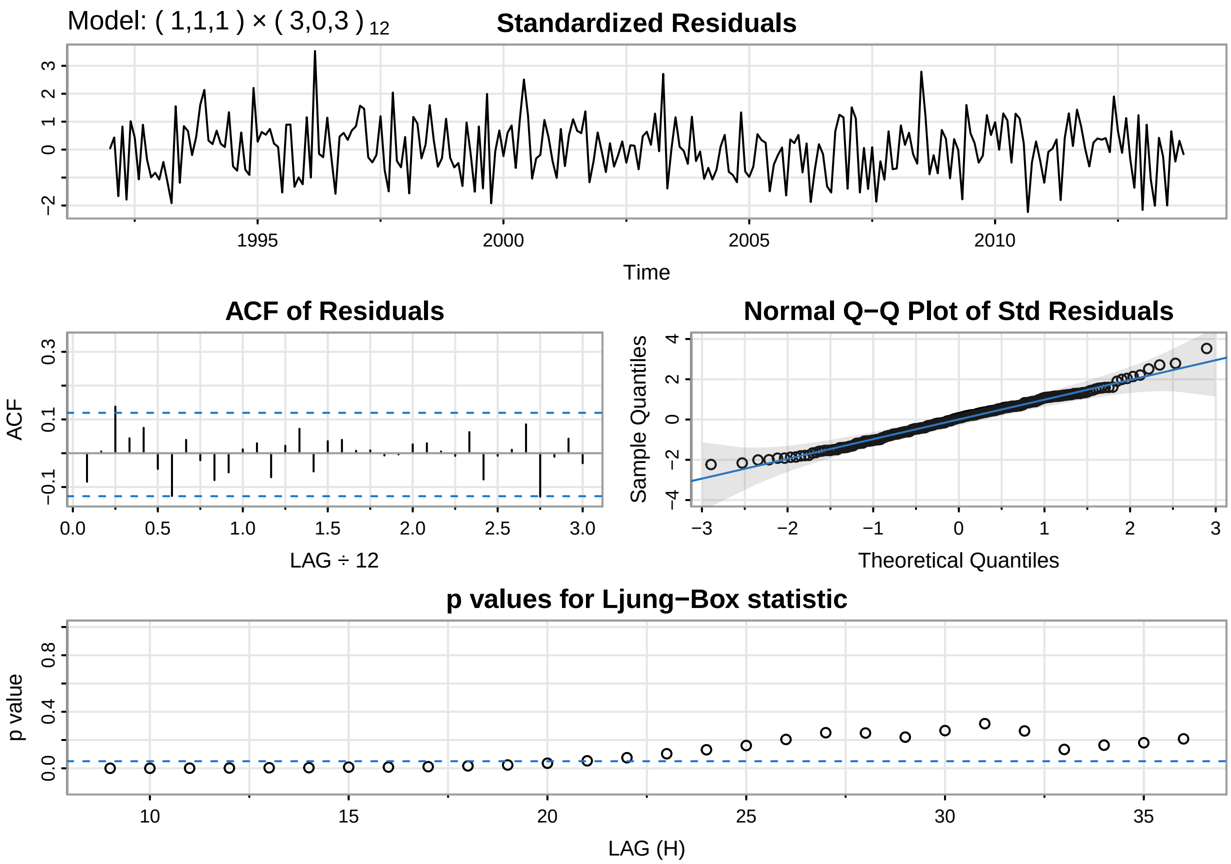
```
## [1] "this gives and AIC of 1159.894608"
```

```r
sys2 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(3,0,3),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 3, D = 0, Q = 3, S = 12)
```

```
## initial  value 1.070751
## iter   2 value 0.837166
## iter   3 value 0.816412
## iter   4 value 0.781708
## iter   5 value 0.779152
## iter   6 value 0.777731
## iter   7 value 0.773011
## iter   8 value 0.766671
## iter   9 value 0.764121
## iter  10 value 0.749060
## iter  11 value 0.729846
## iter  12 value 0.729754
## iter  13 value 0.728821
## iter  14 value 0.723876
## iter  15 value 0.719098
## iter  16 value 0.718459
## iter  17 value 0.717717
## iter  18 value 0.716247
## iter  19 value 0.715679
## iter  20 value 0.715542
## iter  21 value 0.715532
## iter  22 value 0.715509
## iter  23 value 0.715499
## iter  24 value 0.715496
## iter  25 value 0.715488
## iter  26 value 0.715487
## iter  27 value 0.715485
## iter  28 value 0.715483
## iter  29 value 0.715481
## iter  30 value 0.715480
## iter  31 value 0.715478
## iter  32 value 0.715475
## iter  33 value 0.715468
## iter  34 value 0.715458
## iter  35 value 0.715443
## iter  36 value 0.715421
## iter  37 value 0.715391
## iter  38 value 0.715361
## iter  39 value 0.715346
## iter  40 value 0.715346
## iter  41 value 0.715345
## iter  42 value 0.715344
## iter  43 value 0.715344
## iter  44 value 0.715344
## iter  45 value 0.715344
## iter  46 value 0.715344
## iter  47 value 0.715344
## iter  48 value 0.715344
## iter  48 value 0.715344
## iter  48 value 0.715344
## final  value 0.715344
```

```
## converged
## initial  value 0.760193
## iter   2 value 0.757061
## iter   3 value 0.753844
## iter   4 value 0.753492
## iter   5 value 0.752991
## iter   6 value 0.752307
## iter   7 value 0.751285
## iter   8 value 0.750222
## iter   9 value 0.750133
## iter  10 value 0.750063
## iter  11 value 0.749722
## iter  12 value 0.749437
## iter  13 value 0.749137
## iter  14 value 0.748908
## iter  15 value 0.748846
## iter  16 value 0.748808
## iter  17 value 0.748779
## iter  18 value 0.748745
## iter  19 value 0.748722
## iter  20 value 0.748707
## iter  21 value 0.748707
## iter  22 value 0.748705
## iter  23 value 0.748703
## iter  24 value 0.748701
## iter  25 value 0.748697
## iter  26 value 0.748696
## iter  27 value 0.748696
## iter  28 value 0.748690
## iter  29 value 0.748686
## iter  30 value 0.748683
## iter  31 value 0.748682
## iter  32 value 0.748681
## iter  33 value 0.748679
## iter  34 value 0.748677
## iter  35 value 0.748675
## iter  36 value 0.748675
## iter  37 value 0.748675
## iter  38 value 0.748675
## iter  39 value 0.748675
## iter  40 value 0.748675
## iter  40 value 0.748675
## iter  40 value 0.748675
## final  value 0.748675
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE  t.value p.value
## ar1        0.9150 0.0423  21.6307  0.0000
## ma1       -0.7333 0.0695 -10.5573  0.0000
## sar1      -0.2753 0.1538  -1.7900  0.0747
## sar2       0.3021 0.1070   2.8251  0.0051
## sar3       0.7817 0.1215   6.4354  0.0000
```

```
## sma1        0.4775 0.1882   2.5376  0.0118
## sma2        0.1057 0.1372   0.7702  0.4419
## sma3       -0.3950 0.1386  -2.8504  0.0047
## constant   -0.4750 1.6352  -0.2905  0.7717
##
## sigma^2 estimated as 4.217569 on 253 degrees of freedom
##
## AIC = 4.411563  AICc = 4.414289  BIC = 4.547759
##
```




```r
Box.test(residuals(sys2), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys2)
## X-squared = 22.056, df = 20, p-value = 0.3375
```

```r
sprintf("this gives and AIC of %f", sys2$aic)
```

```
## [1] "this gives and AIC of 1153.917850"
```

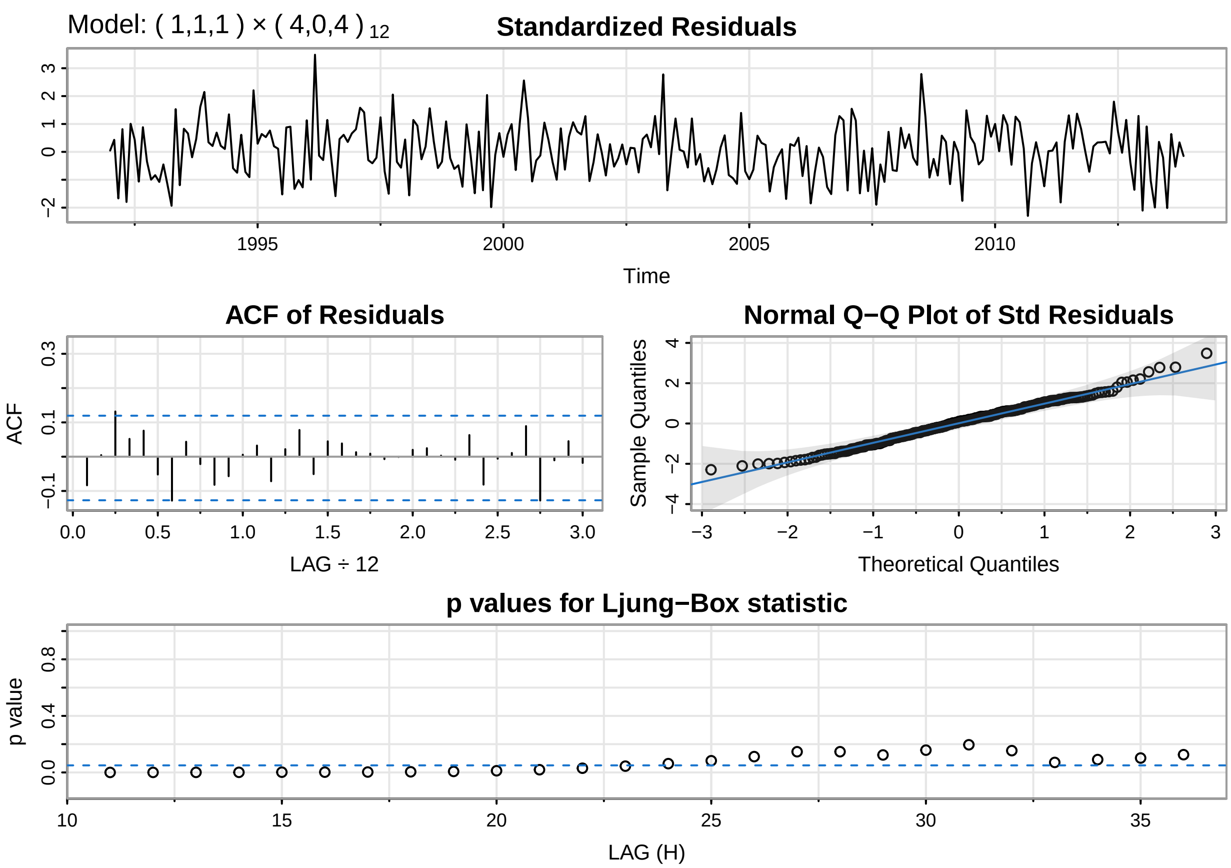
```r
sys3 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(4,0,4),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 4, D = 0, Q = 4, S = 12)
```

```
## initial  value 1.085251
## iter   2 value 0.844277
## iter   3 value 0.807054
## iter   4 value 0.782666
## iter   5 value 0.775299
## iter   6 value 0.773271
## iter   7 value 0.769508
## iter   8 value 0.767341
## iter   9 value 0.766969
## iter  10 value 0.766639
## iter  11 value 0.766177
## iter  12 value 0.765113
## iter  13 value 0.763881
## iter  14 value 0.762516
## iter  15 value 0.762133
## iter  16 value 0.758929
## iter  17 value 0.757338
## iter  18 value 0.750946
## iter  19 value 0.747814
## iter  20 value 0.743774
## iter  21 value 0.740334
## iter  22 value 0.737243
## iter  23 value 0.736106
## iter  24 value 0.735303
## iter  25 value 0.735185
## iter  26 value 0.734927
## iter  27 value 0.734794
## iter  28 value 0.734740
## iter  29 value 0.734653
## iter  30 value 0.734582
## iter  31 value 0.734523
## iter  32 value 0.734513
## iter  33 value 0.734507
## iter  34 value 0.734500
## iter  35 value 0.734495
## iter  36 value 0.734494
## iter  37 value 0.734494
## iter  38 value 0.734494
## iter  39 value 0.734493
## iter  40 value 0.734493
## iter  41 value 0.734491
## iter  42 value 0.734489
## iter  43 value 0.734487
## iter  44 value 0.734485
## iter  45 value 0.734484
## iter  46 value 0.734482
## iter  47 value 0.734479
## iter  48 value 0.734477
## iter  49 value 0.734477
## iter  50 value 0.734476
## iter  51 value 0.734476
```

```
## iter  52 value 0.734475
## iter  53 value 0.734475
## iter  54 value 0.734475
## iter  55 value 0.734474
## iter  56 value 0.734474
## iter  57 value 0.734474
## iter  58 value 0.734474
## iter  59 value 0.734474
## iter  60 value 0.734474
## iter  61 value 0.734474
## iter  62 value 0.734473
## iter  63 value 0.734473
## iter  64 value 0.734473
## iter  65 value 0.734472
## iter  66 value 0.734471
## iter  67 value 0.734471
## iter  68 value 0.734470
## iter  69 value 0.734470
## iter  70 value 0.734470
## iter  71 value 0.734470
## iter  72 value 0.734470
## iter  73 value 0.734470
## iter  74 value 0.734470
## iter  75 value 0.734470
## iter  76 value 0.734470
## iter  77 value 0.734470
## iter  77 value 0.734470
## iter  77 value 0.734470
## final  value 0.734470
## converged
## initial  value 0.760105
## iter   2 value 0.755110
## iter   3 value 0.753379
## iter   4 value 0.752944
## iter   5 value 0.752331
## iter   6 value 0.751374
## iter   7 value 0.750396
## iter   8 value 0.749728
## iter   9 value 0.749551
## iter  10 value 0.749490
## iter  11 value 0.749399
## iter  12 value 0.749204
## iter  13 value 0.749024
## iter  14 value 0.748866
## iter  15 value 0.748749
## iter  16 value 0.748618
## iter  17 value 0.748465
## iter  18 value 0.748368
## iter  19 value 0.748346
## iter  20 value 0.748334
## iter  21 value 0.748326
## iter  22 value 0.748316
## iter  23 value 0.748295
## iter  24 value 0.748272
```

```
## iter  25 value 0.748268
## iter  26 value 0.748265
## iter  27 value 0.748260
## iter  28 value 0.748252
## iter  29 value 0.748236
## iter  30 value 0.748230
## iter  31 value 0.748224
## iter  32 value 0.748214
## iter  33 value 0.748201
## iter  34 value 0.748190
## iter  35 value 0.748183
## iter  36 value 0.748177
## iter  37 value 0.748166
## iter  38 value 0.748150
## iter  39 value 0.748137
## iter  40 value 0.748133
## iter  41 value 0.748132
## iter  42 value 0.748132
## iter  43 value 0.748132
## iter  44 value 0.748131
## iter  45 value 0.748130
## iter  46 value 0.748128
## iter  47 value 0.748127
## iter  48 value 0.748127
## iter  49 value 0.748127
## iter  50 value 0.748127
## iter  51 value 0.748127
## iter  52 value 0.748127
## iter  53 value 0.748127
## iter  54 value 0.748127
## iter  55 value 0.748127
## iter  56 value 0.748126
## iter  57 value 0.748126
## iter  58 value 0.748126
## iter  59 value 0.748125
## iter  60 value 0.748125
## iter  61 value 0.748125
## iter  62 value 0.748125
## iter  63 value 0.748125
## iter  64 value 0.748125
## iter  65 value 0.748124
## iter  66 value 0.748124
## iter  67 value 0.748124
## iter  68 value 0.748124
## iter  69 value 0.748124
## iter  70 value 0.748124
## iter  71 value 0.748124
## iter  72 value 0.748124
## iter  73 value 0.748124
## iter  74 value 0.748124
## iter  75 value 0.748124
## iter  76 value 0.748124
## iter  77 value 0.748123
## iter  78 value 0.748123
```

```
## iter  78 value 0.748123
## iter  78 value 0.748123
## final  value 0.748123
## converged
## <><><><><><><><><><><><><><><><>
##
## Coefficients:
##          Estimate     SE  t.value p.value
## ar1        0.9146 0.0434  21.0527  0.0000
## ma1       -0.7381 0.0717 -10.2901  0.0000
## sar1       0.1422 0.6266   0.2269  0.8207
## sar2       0.3758 0.2427   1.5485  0.1228
## sar3       0.6202 0.2413   2.5703  0.0107
## sar4      -0.2424 0.5186  -0.4674  0.6406
## sma1       0.0634 0.6277   0.1011  0.9196
## sma2      -0.0417 0.3421  -0.1218  0.9032
## sma3      -0.4047 0.1959  -2.0660  0.0399
## sma4       0.0639 0.3187   0.2005  0.8412
## constant  -0.4523 1.6166  -0.2798  0.7799
##
## sigma^2 estimated as 4.217605 on 251 degrees of freedom
##
## AIC = 4.425727  AICc = 4.429758  BIC = 4.589163
##
```



```r
Box.test(residuals(sys3), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys3)
## X-squared = 22.626, df = 20, p-value = 0.3075
```

```r
sprintf("this gives and AIC of %f", sys3$aic)
```

```
## [1] "this gives and AIC of 1157.620841"
```

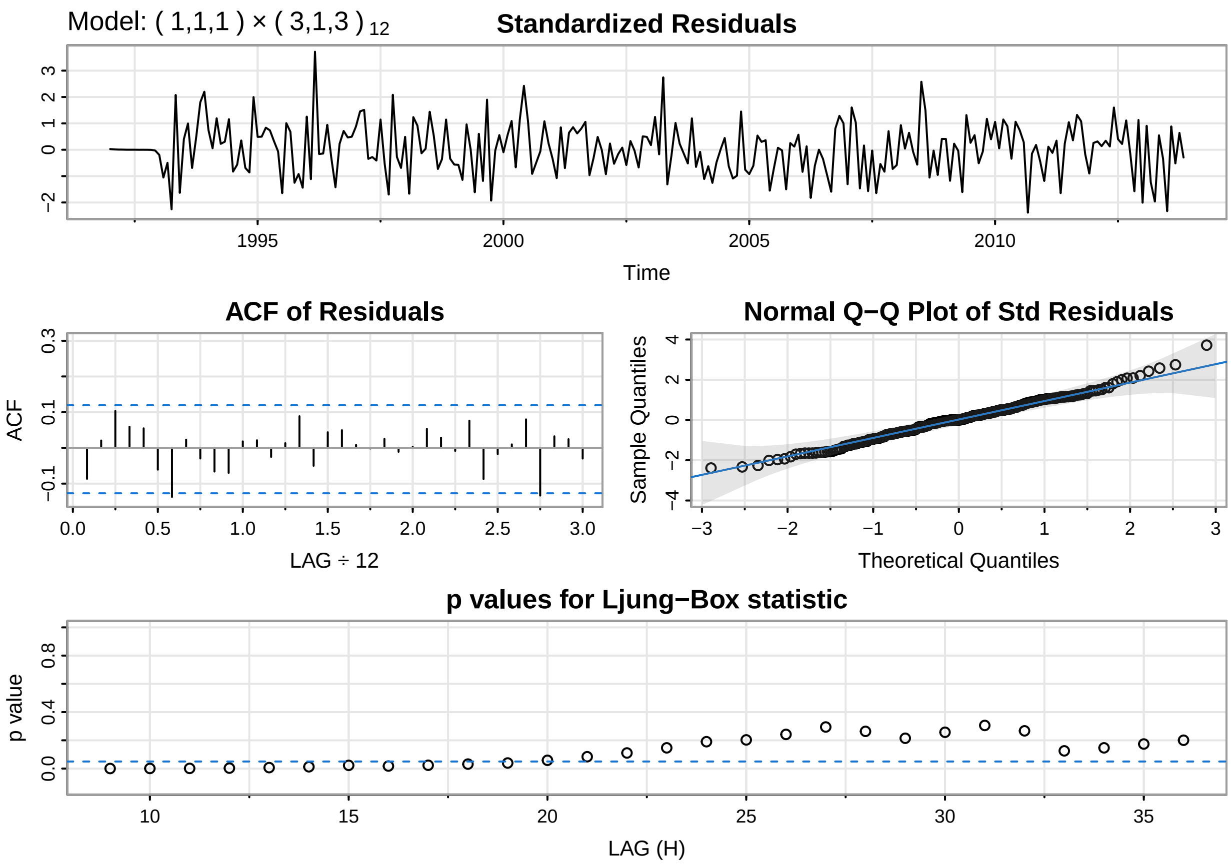
sys2 had the lowest AIC and so I will try a seasonal diferenced fit with those parameters

```
sys4 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(3,1,3),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 3, D = 1, Q = 3, S = 12)
```

```
## initial  value 1.011349
## iter   2 value 0.931875
## iter   3 value 0.816255
## iter   4 value 0.799468
## iter   5 value 0.786906
## iter   6 value 0.779108
## iter   7 value 0.776826
## iter   8 value 0.773687
## iter   9 value 0.772994
## iter  10 value 0.771966
## iter  11 value 0.771761
## iter  12 value 0.771689
## iter  13 value 0.771630
## iter  14 value 0.771337
## iter  15 value 0.767951
## iter  16 value 0.766034
## iter  17 value 0.765474
## iter  18 value 0.761236
## iter  19 value 0.754396
## iter  20 value 0.751200
## iter  21 value 0.746984
## iter  22 value 0.744530
## iter  23 value 0.742656
## iter  24 value 0.741012
## iter  25 value 0.740382
## iter  26 value 0.740044
## iter  27 value 0.739789
## iter  28 value 0.739577
## iter  29 value 0.739458
## iter  30 value 0.739383
## iter  31 value 0.739349
## iter  32 value 0.739334
## iter  33 value 0.739331
## iter  34 value 0.739329
## iter  35 value 0.739327
## iter  36 value 0.739327
## iter  37 value 0.739326
## iter  38 value 0.739326
## iter  39 value 0.739326
## iter  40 value 0.739325
## iter  41 value 0.739325
## iter  42 value 0.739325
## iter  43 value 0.739325
## iter  43 value 0.739325
## iter  43 value 0.739325
## final  value 0.739325
## converged
## initial  value 0.761758
## iter   2 value 0.759776
## iter   3 value 0.758225
```

```
## iter   4 value 0.756904
## iter   5 value 0.755800
## iter   6 value 0.755256
## iter   7 value 0.755117
## iter   8 value 0.754737
## iter   9 value 0.754372
## iter  10 value 0.754301
## iter  11 value 0.754292
## iter  12 value 0.754286
## iter  13 value 0.754260
## iter  14 value 0.754219
## iter  15 value 0.754168
## iter  16 value 0.754150
## iter  17 value 0.754145
## iter  18 value 0.754139
## iter  19 value 0.754138
## iter  20 value 0.754134
## iter  21 value 0.754126
## iter  22 value 0.754118
## iter  23 value 0.754115
## iter  24 value 0.754114
## iter  25 value 0.754113
## iter  26 value 0.754113
## iter  27 value 0.754111
## iter  28 value 0.754109
## iter  29 value 0.754109
## iter  30 value 0.754108
## iter  31 value 0.754107
## iter  32 value 0.754106
## iter  33 value 0.754105
## iter  34 value 0.754104
## iter  35 value 0.754103
## iter  36 value 0.754103
## iter  37 value 0.754102
## iter  38 value 0.754102
## iter  39 value 0.754102
## iter  39 value 0.754102
## iter  39 value 0.754102
## final  value 0.754102
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##      Estimate     SE  t.value p.value
## ar1    0.9266 0.0433  21.4081  0.0000
## ma1   -0.7442 0.0727 -10.2365  0.0000
## sar1  -0.8363 0.4334  -1.9297  0.0548
## sar2  -0.4391 0.4198  -1.0460  0.2966
## sar3   0.1890 0.1717   1.1011  0.2720
## sma1   0.0573 0.4384   0.1308  0.8961
## sma2  -0.0483 0.1684  -0.2868  0.7745
## sma3  -0.4079 0.1488  -2.7419  0.0066
##
## sigma^2 estimated as 4.31914 on 242 degrees of freedom
```

```
##
## AIC = 4.418081  AICc = 4.420471  BIC = 4.544853
##
```




```r
Box.test(residuals(sys4), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys4)
## X-squared = 20.482, df = 20, p-value = 0.4281
```

```r
sprintf("this gives and AIC of %f", sys4$aic)
```

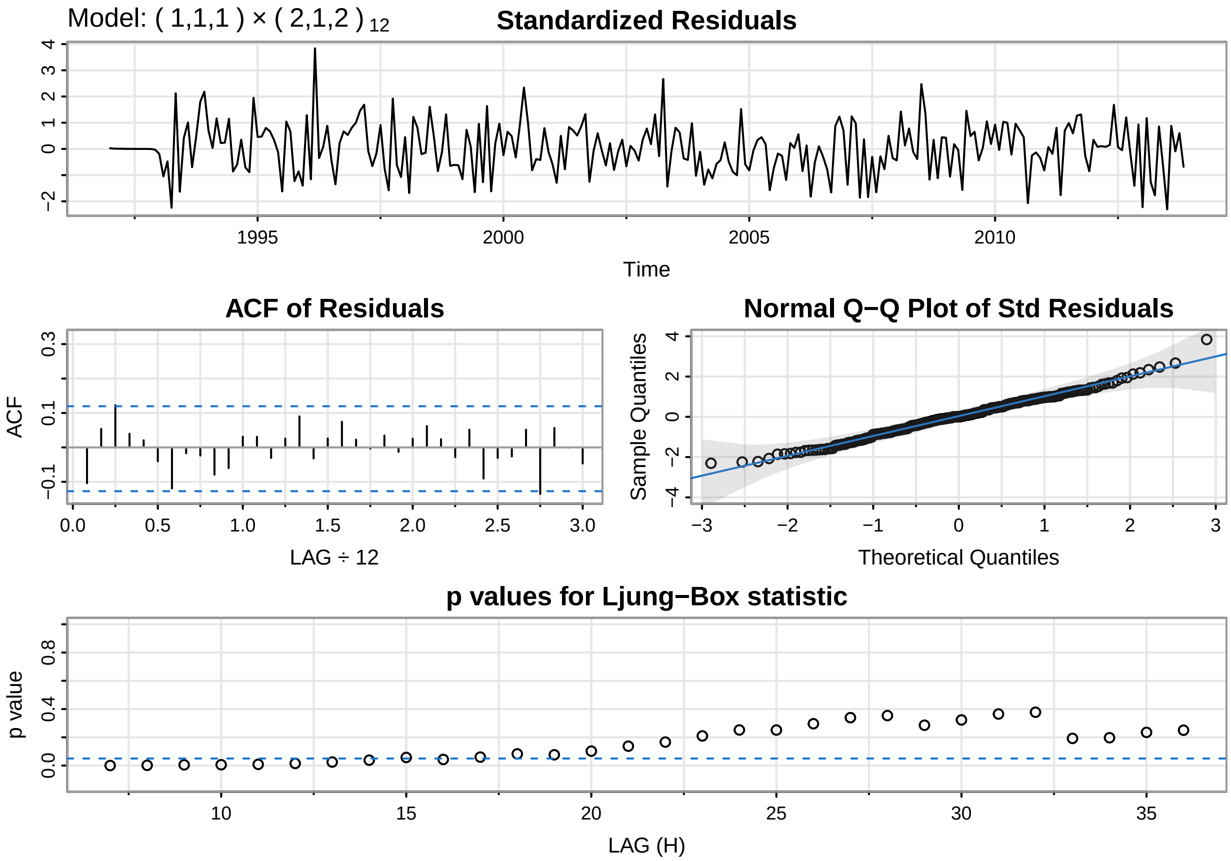
```
## [1] "this gives and AIC of 1104.520218"
```

The AIC greatly improved and the Ljung-Box p values have more values above the dotted line. When I try another seasonal differencing I get an error. Let's try changing P & Q to 2

```r
sys5 <- arima(series.train,order = c(1,1,1),seasonal = list(order = c(2,1,2),period = 12))
sarima(series.train, p = 1, q = 1, d =1, P = 2, D = 1, Q = 2, S = 12)
```

```
## initial  value 1.006413
## iter   2 value 0.903506
## iter   3 value 0.821721
## iter   4 value 0.811880
## iter   5 value 0.805705
## iter   6 value 0.802080
## iter   7 value 0.800291
## iter   8 value 0.787627
## iter   9 value 0.774841
## iter  10 value 0.763281
## iter  11 value 0.758015
## iter  12 value 0.749266
## iter  13 value 0.743215
## iter  14 value 0.738321
## iter  15 value 0.737250
## iter  16 value 0.736836
## iter  17 value 0.736387
## iter  18 value 0.736370
## iter  19 value 0.736364
## iter  20 value 0.736360
## iter  21 value 0.736351
## iter  22 value 0.736341
## iter  23 value 0.736324
## iter  24 value 0.736311
## iter  25 value 0.736308
## iter  26 value 0.736307
## iter  27 value 0.736307
## iter  27 value 0.736307
## final  value 0.736307
## converged
## initial  value 0.777179
## iter   2 value 0.776427
## iter   3 value 0.776097
## iter   4 value 0.776047
## iter   5 value 0.776021
## iter   6 value 0.776016
## iter   7 value 0.776014
## iter   8 value 0.776011
## iter   9 value 0.776002
## iter  10 value 0.776000
## iter  11 value 0.775999
## iter  12 value 0.775999
## iter  13 value 0.775996
## iter  14 value 0.775987
## iter  15 value 0.775978
## iter  16 value 0.775973
## iter  17 value 0.775971
## iter  18 value 0.775962
## iter  19 value 0.775949
## iter  20 value 0.775923
## iter  21 value 0.775874
```

```
## iter  22 value 0.775789
## iter  23 value 0.775681
## iter  24 value 0.775575
## iter  25 value 0.775116
## iter  26 value 0.774566
## iter  27 value 0.774065
## iter  28 value 0.773895
## iter  29 value 0.773756
## iter  30 value 0.773594
## iter  31 value 0.773409
## iter  32 value 0.773339
## iter  33 value 0.773316
## iter  34 value 0.773273
## iter  35 value 0.773224
## iter  36 value 0.773200
## iter  37 value 0.773195
## iter  38 value 0.773192
## iter  39 value 0.773186
## iter  40 value 0.773178
## iter  41 value 0.773175
## iter  42 value 0.773173
## iter  43 value 0.773172
## iter  44 value 0.773171
## iter  45 value 0.773169
## iter  46 value 0.773169
## iter  47 value 0.773168
## iter  48 value 0.773167
## iter  49 value 0.773167
## iter  50 value 0.773167
## iter  51 value 0.773167
## iter  52 value 0.773167
## iter  53 value 0.773167
## iter  54 value 0.773167
## iter  54 value 0.773167
## iter  54 value 0.773167
## final  value 0.773167
## converged
## <><><><><><><><><><><><><><>
##
## Coefficients:
##      Estimate     SE  t.value p.value
## ar1    0.9277 0.0406  22.8491  0.0000
## ma1   -0.7347 0.0700 -10.5002  0.0000
## sar1  -0.8975 0.2408  -3.7269  0.0002
## sar2  -0.2329 0.1198  -1.9436  0.0531
## sma1   0.0973 0.2393   0.4065  0.6847
## sma2  -0.3469 0.1861  -1.8643  0.0635
##
## sigma^2 estimated as 4.537516 on 244 degrees of freedom
##
## AIC = 4.44021  AICc = 4.441593  BIC = 4.538811
##
```




```r
Box.test(residuals(sys5), lag = 20, type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(sys5)
## X-squared = 20.983, df = 20, p-value = 0.3981
```

```r
sprintf("this gives and AIC of %f", sys5$aic)
```

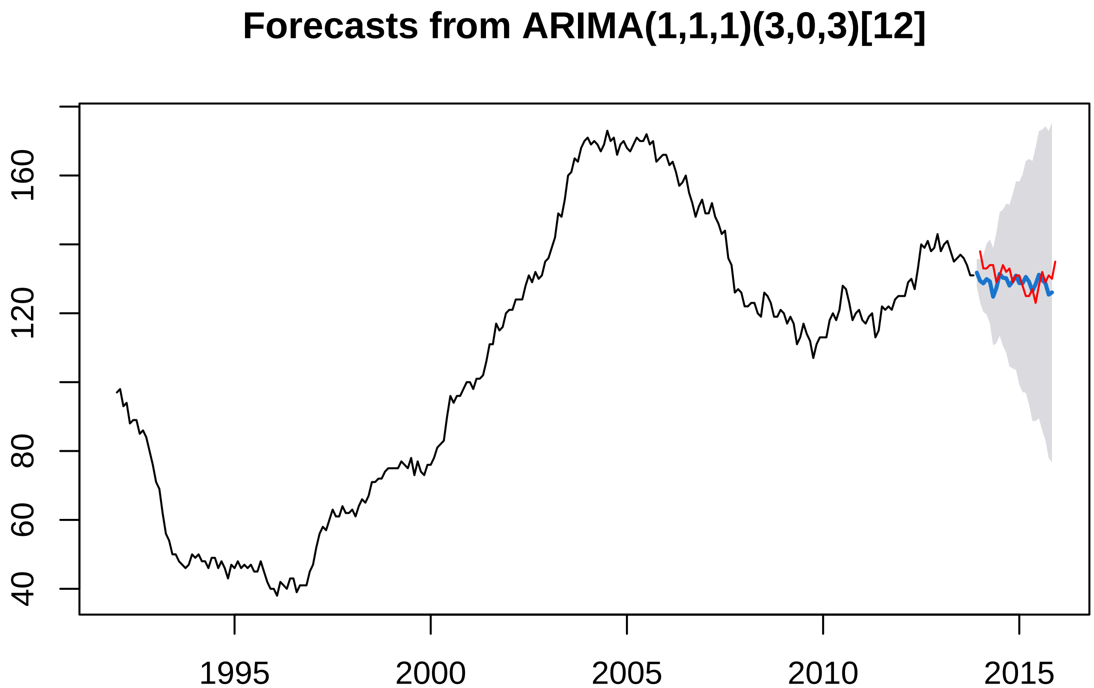
```
## [1] "this gives and AIC of 1110.052539"
```

AIC is not as good as the last one but is definitely still low. Also the Ljung-Box P values chart has greatly improved. I am going to move forward with the following parameters

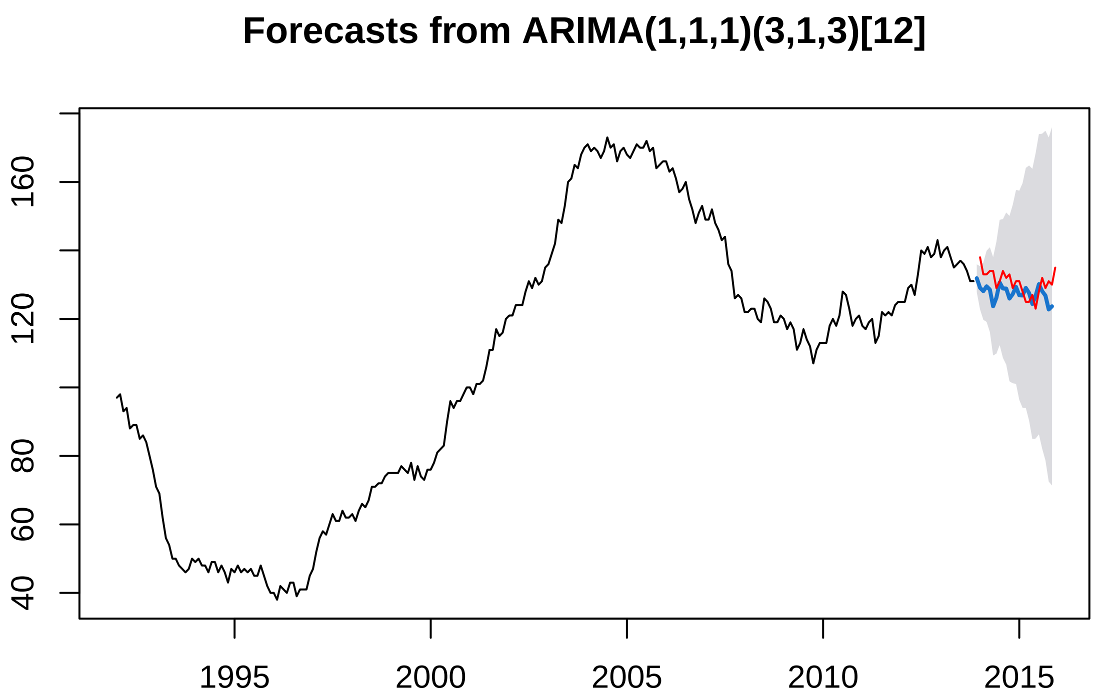
(1,1,1)(3,0,3)[12] (1,1,1)(3,1,3)[12] (1,1,1)(2,1,2)[12]

part (e)

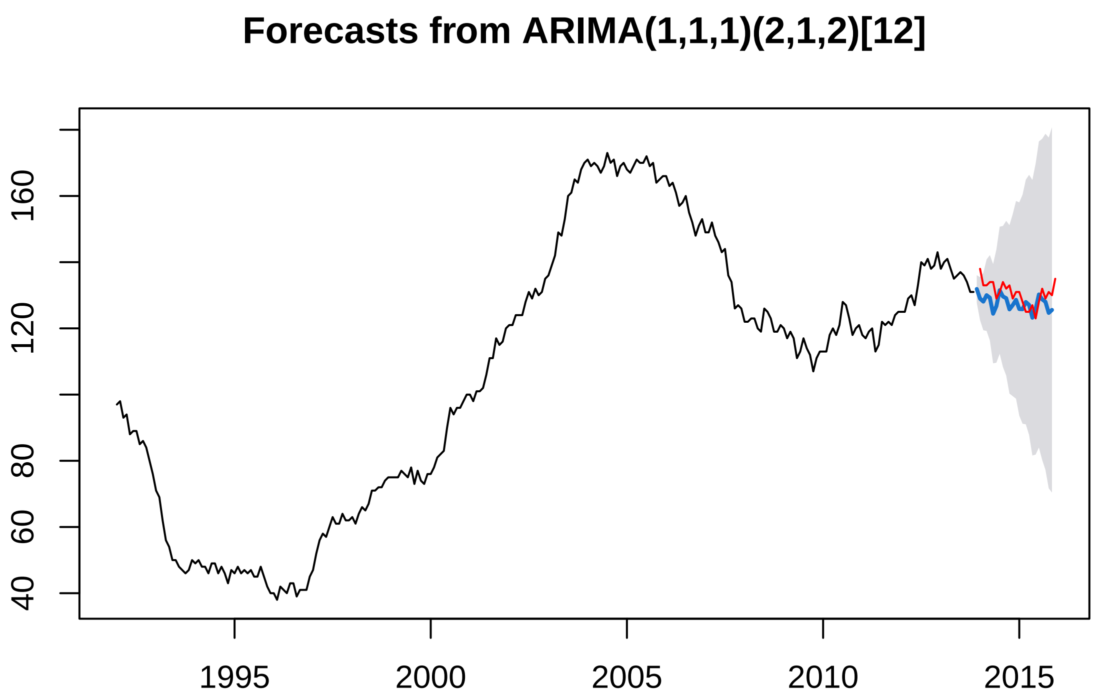
```r
# getting graphics
plot(forecast(sys2, h = 24, level = 0.95))
points(series.val,col='red',type="l")
```




```r
plot(forecast(sys4, h = 24, level = 0.95))
points(series.val,col='red',type="l")
```




```r
plot(forecast(sys5, h = 24, level = 0.95))
points(series.val,col='red',type="l")
```



The prediction interval is much smaller for (1,1,1)(3,0,3)[12]. The distance red and blue trends seems much tighter than the others. Let's see if SSE scores confirm this.

```
forecast(sys2, h = 24, level = 0.95) -> fore1
forecast(sys4, h = 24, level = 0.95) -> fore2
forecast(sys5, h = 24, level = 0.95) -> fore3

sum((fore1$mean - series.val)^2)
```

```
## [1] 399.7853
```

```
sum((fore2$mean - series.val)^2)
```

```
## [1] 545.6536
```

```
sum((fore3$mean - series.val)^2)
```

```
## [1] 466.6076
```

This confirms my intuition. ARIMA(1,1,1)(3,0,3)[12] performs the best out of all three models. I believe it is in the companies best interest to use this model for future analysis because of how much better it preforms compared to all the other models test.