

Univariate Forecasts

In this part we evaluate univariate forecasts of the gas consumption.

Graphical Analysis and stationarity

Load useful packages & data

```
library(readr)
library(forecast)

## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##      as.Date, as.Date.numeric
## Loading required package: timeDate
## This is forecast 7.3
library(ggplot2)
library(knitr)
gas <- na.omit(read_csv("data/gas.csv", col_types = cols(date = col_date(format = "%Y-%m-%d"))))
```

We also define functions for the RMS error which we will use as the main evaluation metric here.

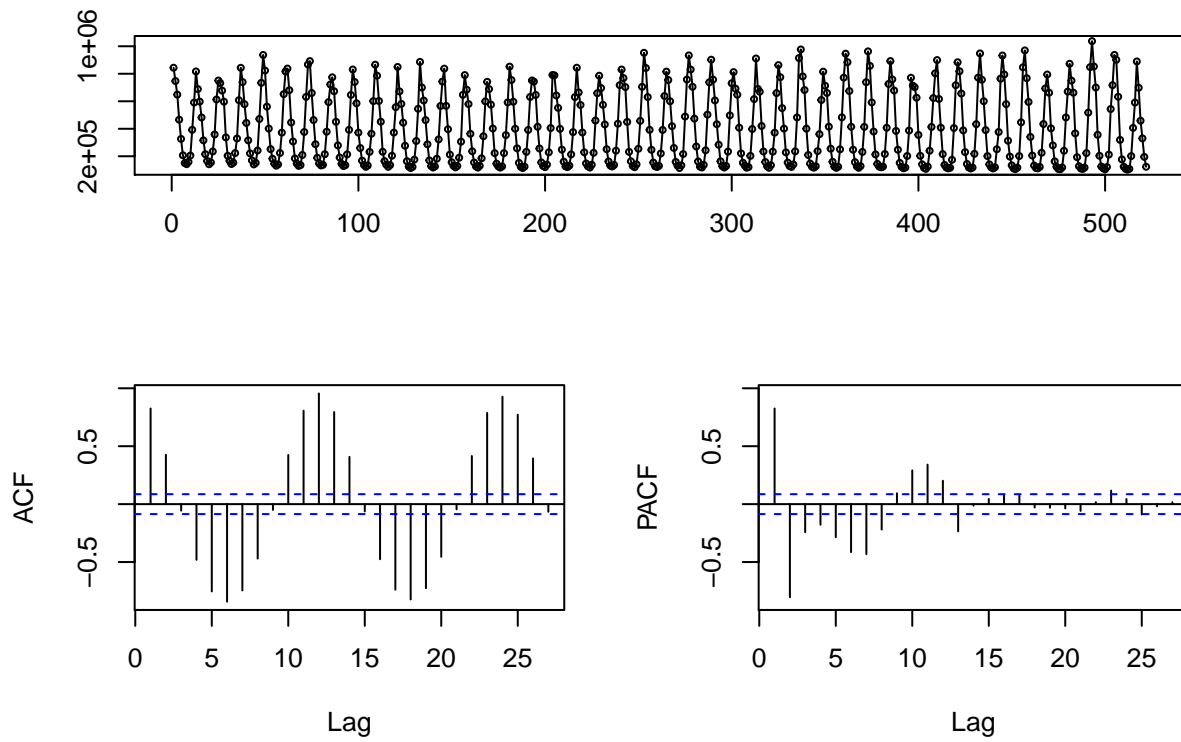
```
direct_rmse <- function(l, r) {sqrt(sum((l - r)^2)/NROW(l))}
exp_rmse <- function(l, r) {direct_rmse(exp(l), exp(r))}
```

Stationarity and integration of the consumption

Let us plot the values, ACF and PACF of the data and its logarithm, with seasonal and annual differentiation.

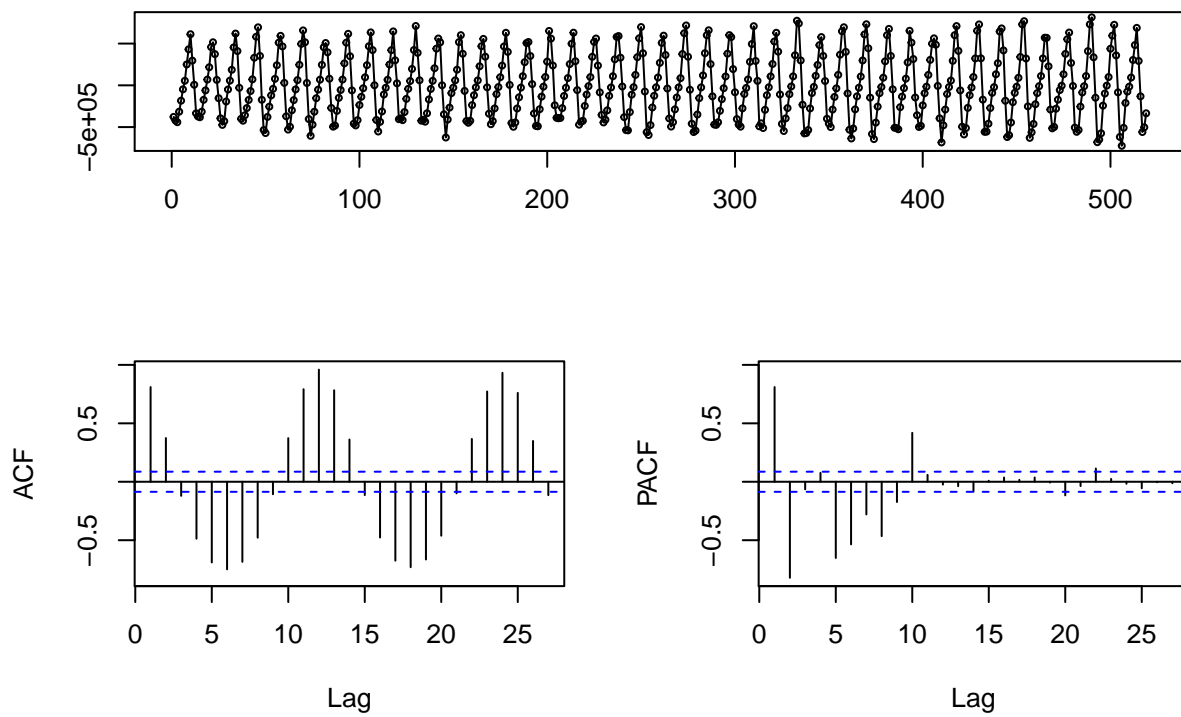
```
tsdisplay(gas$gas_cons, main="Gas Consumption")
```

Gas Consumption

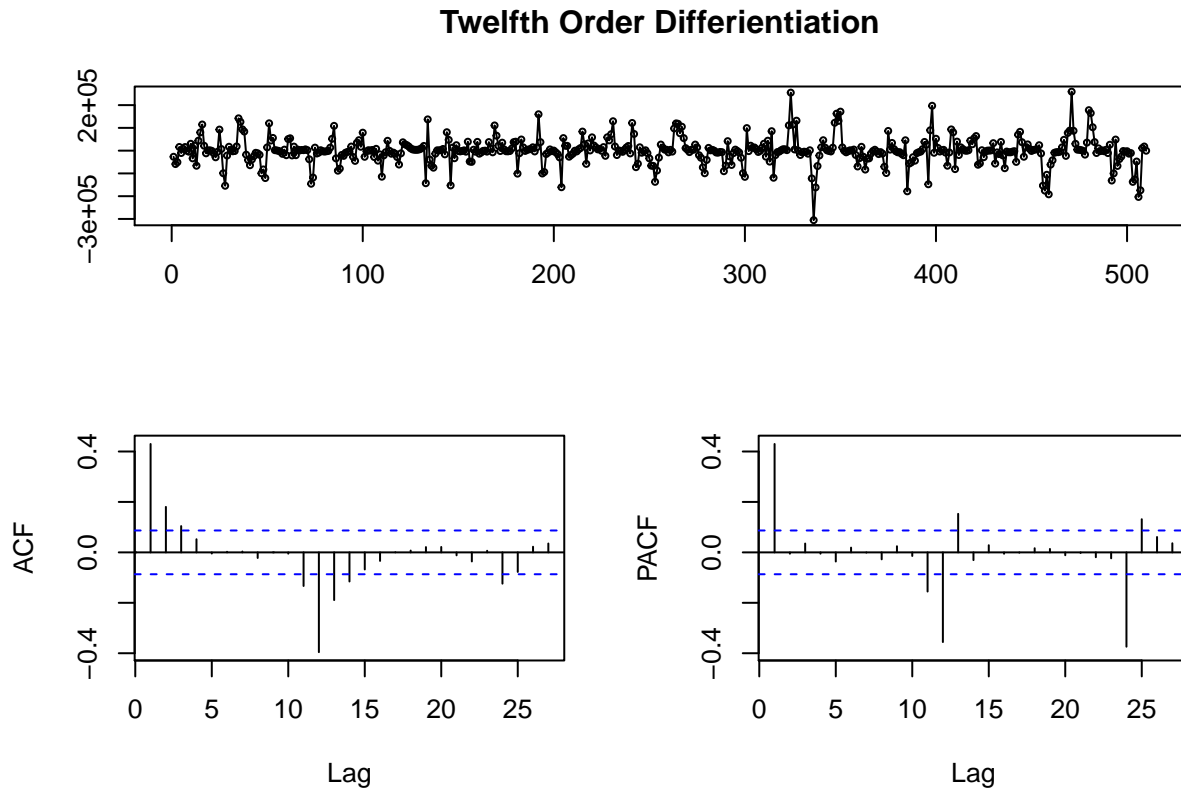


```
tsdisplay(diff(gas$gas_cons, 3), main="Third Order Differentiation")
```

Third Order Differentiation



```
tsdisplay(diff(gas$gas_cons, 12), main="Twelfth Order Differentiation")
```

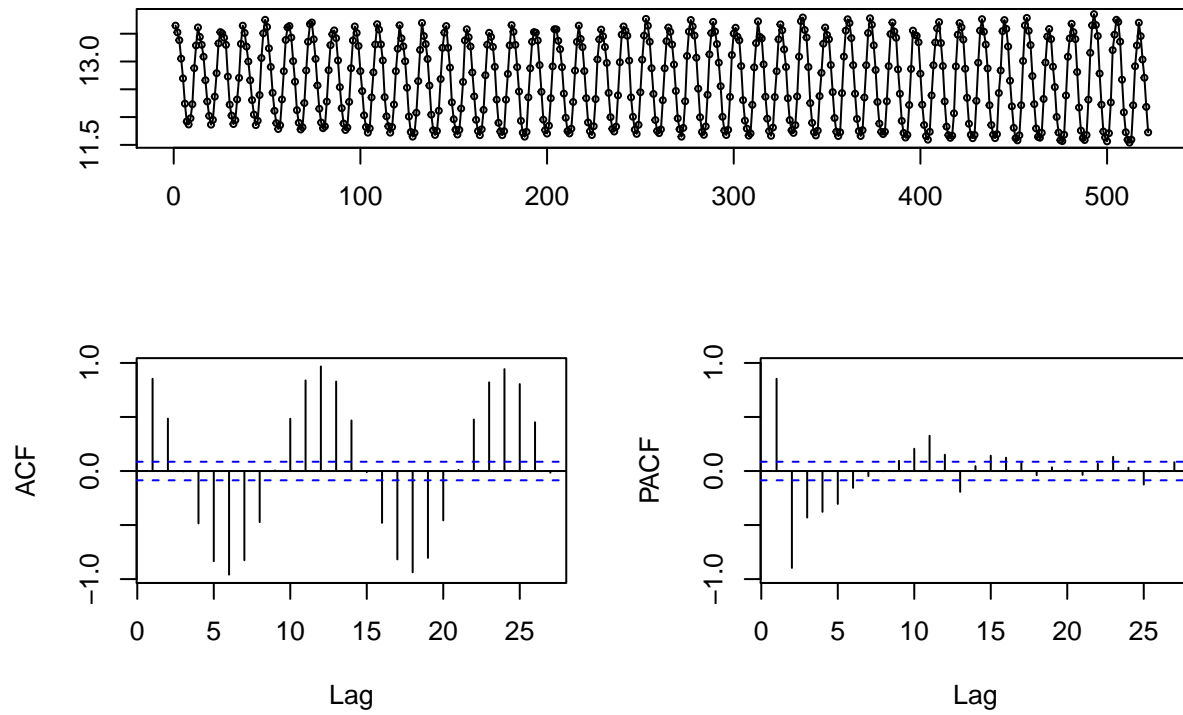


The results are similar to that of the log of the consumption. Annual differencing seem to yield a somewhat stationary process with no obvious seasonality.

Stationarity and integration of the log of the consumption

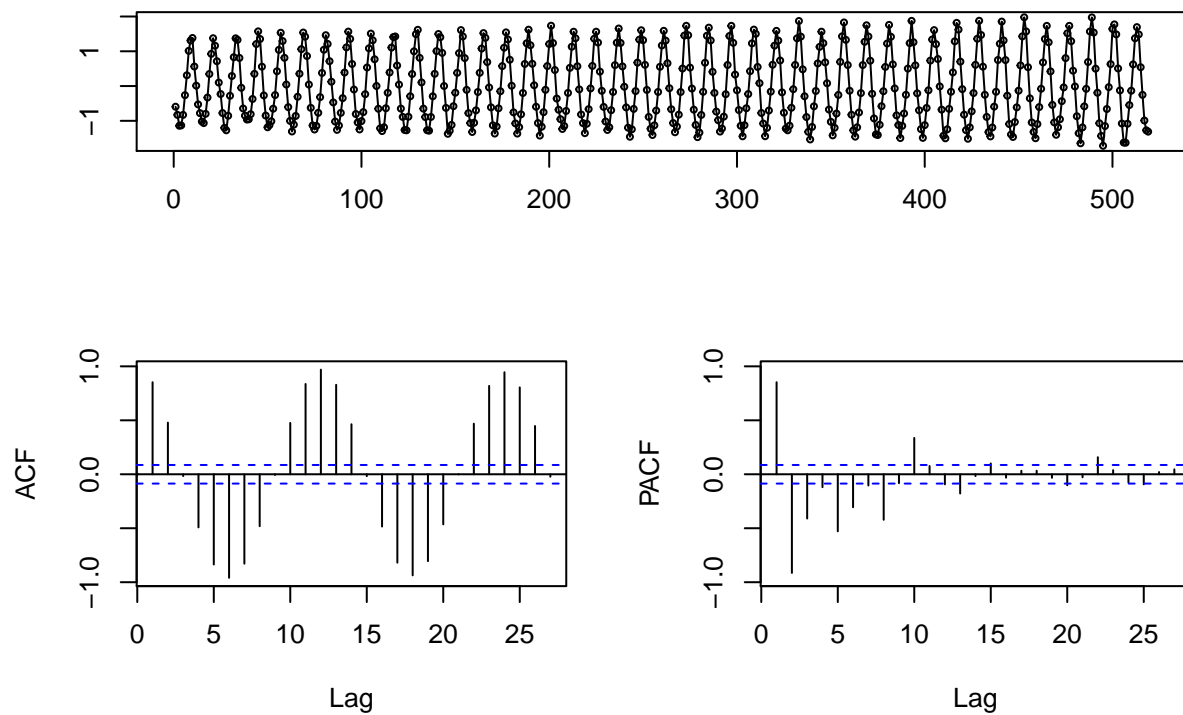
```
tsdisplay(gas$log_gas_cons,main="Log transform of Gas Consumption")
```

Log transform of Gas Consumption

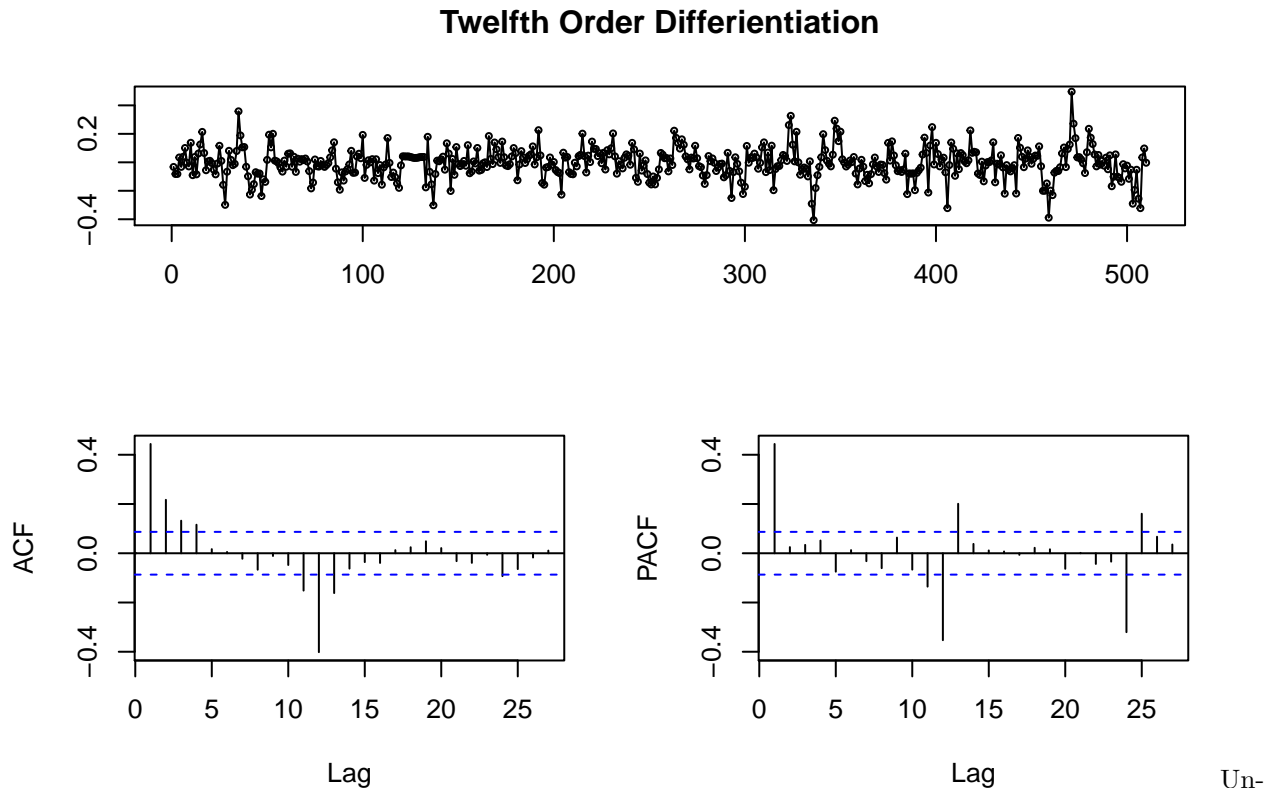


```
tsdisplay(diff(gas$log_gas_cons, 3), main="Third Order Differentiation")
```

Third Order Differentiation



```
tsdisplay(diff(gas$log_gas_cons, 12), main="Twelfth Order Differentiation")
```



Un-
surprisingly, the results are globally similar: we notice a strong seasonality, still observable after seasonal differencing. Exploring further the dynamics of the log consumption with annual differencing yields a time series that looks stationary.

Let us perform statistical tests on the log gas consumption for stationarity with differencing and seasonal differencing:

```
ndiffs(gas$log_gas_cons, alpha=0.05, test=c("kpss", "adf", "pp"), max.d=2)
```

```
## [1] 0
```

```
nsdiffs(gas$log_gas_cons, m=12, test=c("ocsb", "ch"), max.D=1)
```

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

The unit root test with level 0.05 indicates there is no need for differencing to obtain a stationary time series, but here the strong seasonality makes it irrelevant. We thus look at seasonal differencing which shows there is a need for first order seasonal differencing for the series to become stationary.

Pre-processing

Let us convert data into time series and split it into a training set and a one year test set to evaluate our first models.

```
lgas = ts(gas$log_gas_cons, start = c(1973, 1), frequency=12)
train = window(lgas, start = c(1973, 1), end = c(2015, 6))
test = window(lgas, start = c(2015, 7), end = c(2016, 6))

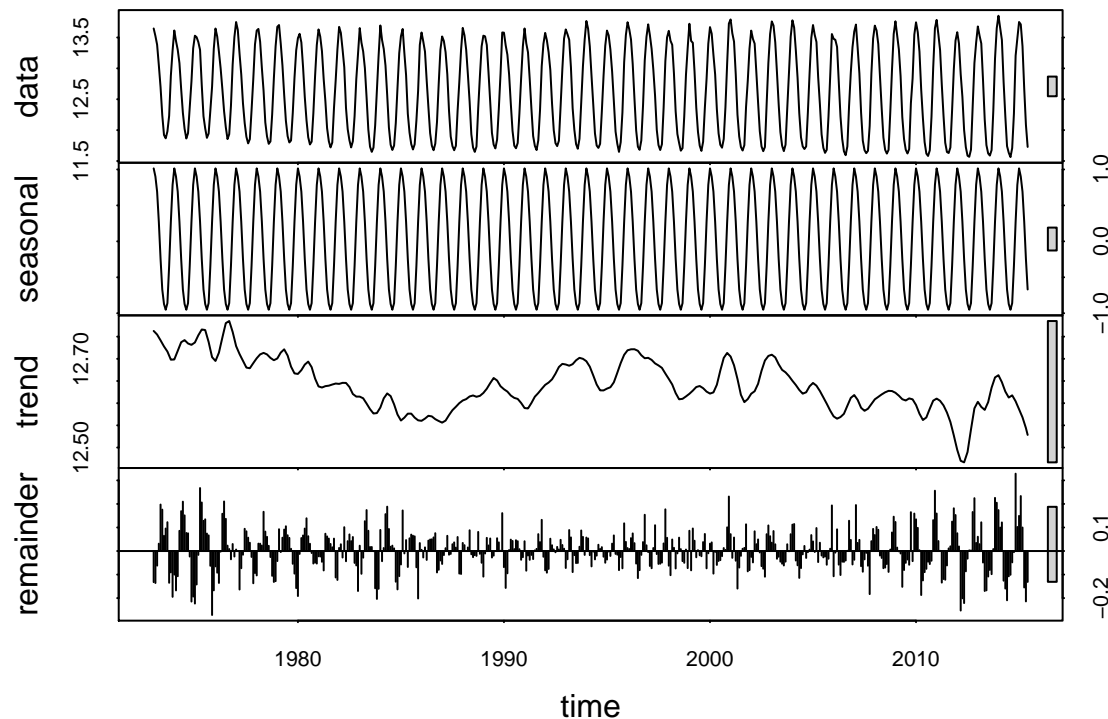
dgas = ts(gas$gas_cons, start = c(1973, 1), frequency=12)
```

```
dtrain = window(dgas, start = c(1973, 1), end = c(2015, 6))
dtest = window(dgas, start = c(2015, 7), end = c(2016, 6))
```

Time Series Decomposition with log gas consumption

Let's try to decompose our series in three components: trend, seasonality and error. We assume seasonality is fixed across years and we do our analysis on the log of consumption data to avoid overweighing extreme values. This is a strong assumption that is equivalent to doing a multiplicative decomposition.

```
decomp = stl(train, s.window="periodic")
plot(decomp)
```



The stl decomposition seems to capture well the seasonality “graphically”. We’ll see how well this translates into forecasts. Our first attempt will just forecast the time series by removing seasonality, and then using the last observation to which we add back the seasonality as the next forecast value.

```
stlfc_naive = forecast(decomp, method="naive", h=12)
summary(stlfc_naive)
```

```
##
## Forecast method: STL + Random walk
##
## Model Information:
## $drift
## [1] 0
##
## $drift.se
## [1] 0
##
```

```
## $sd
## [1] 0.09023216
##
## $call
## rwf(y = x, h = h, drift = FALSE, level = level)
##
##
## Error measures:
##           ME           RMSE           MAE           MPE           MAPE
## Training set -0.0004587216 0.09014465 0.06759804 -0.006234602 0.5284953
##           MASE           ACF1
## Training set 0.8732574 -0.09577716
##
## Forecasts:
##           Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
## Jul 2015           11.52510 11.40957 11.64062 11.34842 11.70178
## Aug 2015           11.44640 11.33087 11.56192 11.26972 11.62308
## Sep 2015           11.53739 11.42186 11.65291 11.36071 11.71407
## Oct 2015           12.05578 11.94026 12.17131 11.87910 12.23246
## Nov 2015           12.67021 12.55469 12.78574 12.49353 12.84690
## Dec 2015           13.18247 13.06695 13.29800 13.00579 13.35915
## Jan 2016           13.41230 13.29677 13.52782 13.23562 13.58898
## Feb 2016           13.29855 13.18303 13.41408 13.12187 13.47523
## Mar 2016           13.08706 12.97154 13.20259 12.91038 13.26374
## Apr 2016           12.66947 12.55395 12.78500 12.49279 12.84615
## May 2016           12.15371 12.03818 12.26923 11.97703 12.33039
## Jun 2016           11.72784 11.61231 11.84336 11.55115 11.90452
```

```
accuracy(stlfc_naive, test)
```

```
##           ME           RMSE           MAE           MPE           MAPE
## Training set -0.0004587216 0.09014465 0.06759804 -0.006234602 0.5284953
## Test set      0.0984665485 0.13661429 0.10752662  0.769520343 0.8394485
##           MASE           ACF1 Theil's U
## Training set 0.8732574 -0.09577716      NA
## Test set      1.3890700 0.29228001 0.3240292
```

Let's try with exponential smoothing to forecast the seasonally-adjusted series.

```
stlfc_ets = forecast(decomp, method="ets", h=12)
summary(stlfc_ets)
```

```
##
## Forecast method: STL + ETS(M,N,N)
##
## Model Information:
## ETS(M,N,N)
##
## Call:
## ets(y = x, model = etsmodel, allow.multiplicative.trend = allow.multiplicative.trend)
##
## Smoothing parameters:
##   alpha = 0.8899
##
## Initial states:
##   l = 12.6302
```

```
##
##   sigma: 0.0071
##
##       AIC       AICc       BIC
## 724.4449 724.4923 737.1481
##
## Error measures:
##               ME       RMSE       MAE       MPE       MAPE
## Training set -0.0005270008 0.08957449 0.06699331 -0.00705005 0.524313
##               MASE       ACF1
## Training set 0.8654452 0.006014702
##
## Forecasts:
##      Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jul 2015      11.51893 11.40642 11.63144 11.34687 11.69100
## Aug 2015      11.44023 11.28963 11.59084 11.20990 11.67056
## Sep 2015      11.53122 11.35037 11.71207 11.25464 11.80781
## Oct 2015      12.04961 11.84290 12.25633 11.73347 12.36576
## Nov 2015      12.66405 12.43436 12.89374 12.31277 13.01532
## Dec 2015      13.17631 12.92574 13.42687 12.79310 13.55951
## Jan 2016      13.40613 13.13630 13.67596 12.99346 13.81880
## Feb 2016      13.29238 13.00458 13.58019 12.85222 13.73255
## Mar 2016      13.08090 12.77616 13.38563 12.61485 13.54694
## Apr 2016      12.66331 12.34254 12.98407 12.17274 13.15387
## May 2016      12.14754 11.81151 12.48357 11.63363 12.66146
## Jun 2016      11.72167 11.37103 12.07230 11.18542 12.25792
```

```
accuracy(stlfc_ets, test)
```

```
##               ME       RMSE       MAE       MPE       MAPE
## Training set -0.0005270008 0.08957449 0.06699331 -0.00705005 0.5243130
## Test set      0.1046336198 0.14112405 0.11216904 0.81905384 0.8768599
##               MASE       ACF1 Theil's U
## Training set 0.8654452 0.006014702      NA
## Test set      1.4490426 0.292280013 0.3346012
```

It actually does worse than the naive technique. As a last attempt, we can also use arima on the seasonally-adjusted data.

```
stlfc_arima = forecast(decomp, method="arima", h=12)
summary(stlfc_arima)
```

```
##
## Forecast method: STL + ARIMA(3,1,2)
##
## Model Information:
## Series: x
## ARIMA(3,1,2)
##
## Coefficients:
##      ar1      ar2      ar3      ma1      ma2
##      1.3257 -0.4219 -0.1728 -1.6815 0.7146
## s.e. 0.0619 0.0805 0.0491 0.0466 0.0483
##
## sigma^2 estimated as 0.006182: log likelihood=573.58
## AIC=-1135.15 AICc=-1134.99 BIC=-1109.76
```



```
##
## Error measures:
##           ME           RMSE           MAE           MPE           MAPE
## Training set -0.00173105 0.07816114 0.0588041 -0.01744146 0.4599125
##           MASE           ACF1
## Training set 0.7596539 -0.02219796
##
## Forecasts:
##           Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
## Jul 2015           11.60443 11.50366 11.70519 11.45032 11.75853
## Aug 2015           11.61868 11.49882 11.73854 11.43537 11.80199
## Sep 2015           11.78782 11.65912 11.91652 11.59099 11.98465
## Oct 2015           12.35688 12.22653 12.48724 12.15752 12.55624
## Nov 2015           12.98946 12.85910 13.11981 12.79010 13.18882
## Dec 2015           13.49089 13.35981 13.62197 13.29042 13.69136
## Jan 2016           13.68995 13.55758 13.82232 13.48751 13.89239
## Feb 2016           13.53685 13.40361 13.67009 13.33308 13.74063
## Mar 2016           13.28805 13.15464 13.42146 13.08402 13.49208
## Apr 2016           12.84291 12.70944 12.97638 12.63878 13.04703
## May 2016           12.31316 12.17894 12.44739 12.10788 12.51844
## Jun 2016           11.88682 11.75087 12.02277 11.67890 12.09474
```

```
accuracy(stlfc_arima, test)
```

```
##           ME           RMSE           MAE           MPE           MAPE
## Training set -0.00173105 0.07816114 0.0588041 -0.01744146 0.4599125
## Test set      -0.12150070 0.14220379 0.1228988 -0.97753194 0.9877384
##           MASE           ACF1 Theil's U
## Training set 0.7596539 -0.02219796      NA
## Test set      1.5876540 -0.15231347 0.3438054
```

R picked an ARIMA(3,1,2). The results aren't as good as with the naive approach either. To conclude on this approach, it seems that the best model is to do a naive forecast on the seasonally-adjusted data.

We should note that what we are interested in is the RMSE on the test set for the values of the gas consumption, and not for their log.

```
kable(data.frame("naive"=exp_rmse(stlfc_naive$mean, test),
                 "ets"=exp_rmse(stlfc_ets$mean, test),
                 "arima"=exp_rmse(stlfc_arima$mean, test), row.names = "RMSE"), align='c')
```

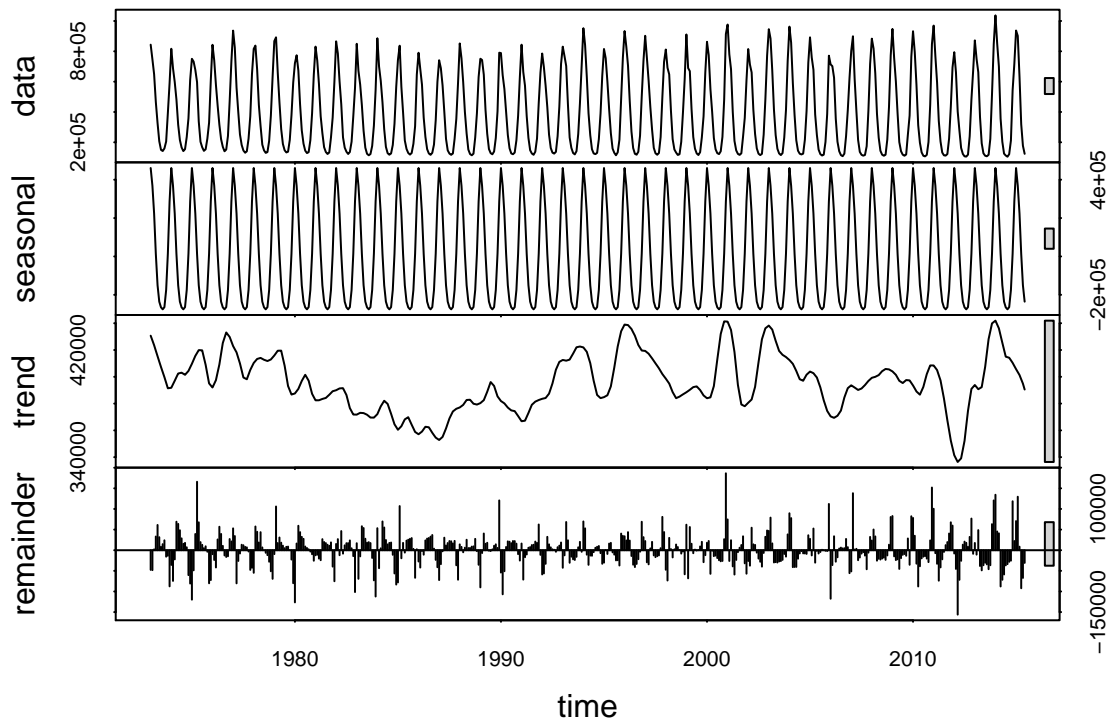
	naive	ets	arima
RMSE	77722.56	79507.5	60795.39

According to this criterion, the model to pick is the ARIMA one.

Time Series Decomposition with direct gas consumption

The approach is the same as above, but this time we fit the models to the data directly and not to the log of it. It is equivalent to doing an additive decomposition.

```
ddecomp = stl(dtrain, s.window="periodic")
plot(ddecomp)
```



Just as in the previous case, graphically the decomposition seems interesting. The trend seems a bit harder to spot in this one. There seems to be extreme peaks which may be problematic for our forecasts.

Let's fit the naive, exponential smoothing and arima models now.

```
dstl_naive = forecast(ddecomp, method="naive", h=12)
dstl_ets = forecast(ddecomp, method="ets", h=12)
dstl_arima = forecast(ddecomp, method="arima", h=12)

summary(dstl_naive)

##
## Forecast method: STL + Random walk
##
## Model Information:
## $drift
## [1] 0
##
## $drift.se
## [1] 0
##
## $sd
## [1] 48015.64
##
## $call
## rwf(y = x, h = h, drift = FALSE, level = level)
##
##
## Error measures:
##           ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -45.75346 47968.48 31623.02 -0.2747371 7.751035 0.8744243
##           ACF1
```

```
## Training set -0.1915393
```

```
##
```

```
## Forecasts:
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## Jul 2015	94270.84	32796.76	155744.9	254.3545	188287.3
## Aug 2015	84031.36	22557.28	145505.4	-9985.1258	178047.8
## Sep 2015	95367.62	33893.54	156841.7	1351.1316	189384.1
## Oct 2015	184854.82	123380.74	246328.9	90838.3333	278871.3
## Nov 2015	374232.89	312758.81	435707.0	280216.4028	468249.4
## Dec 2015	646363.90	584889.82	707838.0	552347.4122	740380.4
## Jan 2016	820611.49	759137.42	882085.6	726595.0074	914628.0
## Feb 2016	728237.75	666763.68	789711.8	634221.2658	822254.2
## Mar 2016	582575.01	521100.94	644049.1	488558.5297	676591.5
## Apr 2016	372965.53	311491.45	434439.6	278949.0453	466982.0
## May 2016	209496.61	148022.54	270970.7	115480.1311	303513.1
## Jun 2016	123975.00	62500.93	185449.1	29958.5166	217991.5

```
summary(dstl_ets)
```

```
##
```

```
## Forecast method: STL + ETS(A,N,N)
```

```
##
```

```
## Model Information:
```

```
## ETS(A,N,N)
```

```
##
```

```
## Call:
```

```
## ets(y = x, model = etsmodel, allow.multiplicative.trend = allow.multiplicative.trend)
```

```
##
```

```
## Smoothing parameters:
```

```
## alpha = 0.6914
```

```
##
```

```
## Initial states:
```

```
## l = 419740.7679
```

```
##
```

```
## sigma: 46595.57
```

```
##
```

```
## AIC AICc BIC
```

```
## 14149.80 14149.84 14162.50
```

```
##
```

```
## Error measures:
```

	ME	RMSE	MAE	MPE	MAPE	MASE
--	----	------	-----	-----	------	------

## Training set	-192.7531	46595.57	31620.76	-0.4278733	8.156581	0.8743619
-----------------	-----------	----------	----------	------------	----------	-----------

```
## ACF1
```

```
## Training set 0.05871077
```

```
##
```

```
## Forecasts:
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
## Jul 2015	86296.34	26581.714	146011.0	-5029.299	177622.0
## Aug 2015	76056.86	3459.412	148654.3	-34971.356	187085.1
## Sep 2015	87393.12	3876.996	170909.2	-40333.768	215120.0
## Oct 2015	176880.32	83716.509	270044.1	34398.567	319362.1
## Nov 2015	366258.39	264356.237	468160.5	210412.497	522104.3
## Dec 2015	638389.40	528441.222	748337.6	470238.173	806540.6
## Jan 2016	812636.99	695192.732	930081.3	633021.496	992252.5
## Feb 2016	720263.25	595773.464	844753.0	529872.550	910654.0

```
## Mar 2016      574600.52 443443.128 705757.9 374012.601 775188.4
## Apr 2016      364991.03 227488.983 502493.1 154699.795 575282.3
## May 2016      201522.12 57955.525 345088.7 -18044.038 421088.3
## Jun 2016      116000.50 -33384.635 265385.6 -112464.350 344465.4
```

```
summary(dstl_arima)
```

```
##
## Forecast method: STL + ARIMA(1,0,0) with non-zero mean
##
## Model Information:
## Series: x
## ARIMA(1,0,0) with non-zero mean
##
## Coefficients:
##          ar1  intercept
##          0.5363 395398.318
## s.e.    0.0373   4003.951
##
## sigma^2 estimated as 1.773e+09: log likelihood=-6153.29
## AIC=12312.57   AICc=12312.62   BIC=12325.28
##
## Error measures:
##              ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -19.92559 42023.99 27299.12 -0.7325736 6.236539 0.7548619
##              ACF1
## Training set 0.02936418
##
## Forecasts:
##      Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
## Jul 2015      110802.1  56840.27 164763.9  28274.61 193329.6
## Aug 2015      109428.1  48196.21 170660.0  15782.00 203074.2
## Sep 2015      125518.8  62350.75 188686.9  28911.60 222126.0
## Oct 2015      217555.8 153841.75 281269.8 120113.59 314997.9
## Nov 2015      408301.2 344431.06 472171.4 310620.24 505982.2
## Dec 2015      681165.6 617250.55 745080.6 583415.99 778915.1
## Jan 2016      855806.4 791878.52 919734.3 758037.14 953575.7
## Feb 2016      763643.6 699711.98 827575.2 665868.63 861418.5
## Mar 2016      618094.0 554161.28 682026.6 520317.37 715870.5
## Apr 2016      408545.1 344612.15 472478.1 310768.07 506322.2
## May 2016      245108.7 181175.68 309041.8 147331.55 342885.9
## Jun 2016      159604.6 95671.48 223537.7 61827.35 257381.8
```

```
accuracy(dstl_naive, dtest)
```

```
##              ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -45.75346 47968.48 31623.02 -0.2747371 7.751035 0.8744243
## Test set     -8647.06715 48462.13 35825.07 0.4779919 10.670010 0.9906173
##              ACF1 Theil's U
## Training set -0.19153932      NA
## Test set     0.07630088 0.2475476
```

```
accuracy(dstl_arima, dtest)
```

```
##              ME      RMSE      MAE      MPE      MAPE
## Training set -19.92559 42023.99 27299.12 -0.7325736 6.236539
```

```
## Test set      -40862.99755  64275.22  46467.43  -13.9318194  14.561932
##              MASE          ACF1 Theil's U
## Training set  0.7548619  0.02936418      NA
## Test set      1.2848947  0.12975328  0.2997268
```

```
accuracy(dstl_ets, dtest)
```

```
##              ME      RMSE      MAE      MPE      MAPE      MASE
## Training set -192.7531 46595.57 31620.76 -0.4278733  8.156581  0.8743619
## Test set     -672.5696 47689.19 37092.65  4.3520340 12.707022  1.0256679
##              ACF1 Theil's U
## Training set  0.05871077      NA
## Test set      0.07630088  0.3058589
```

This time an ARIMA(1, 0, 0) is picked. The best performing model is the exponential smoothing one (ets). This additive approach performs better than the multiplicative approach we outlined in the previous section.

Seasonal ARIMA

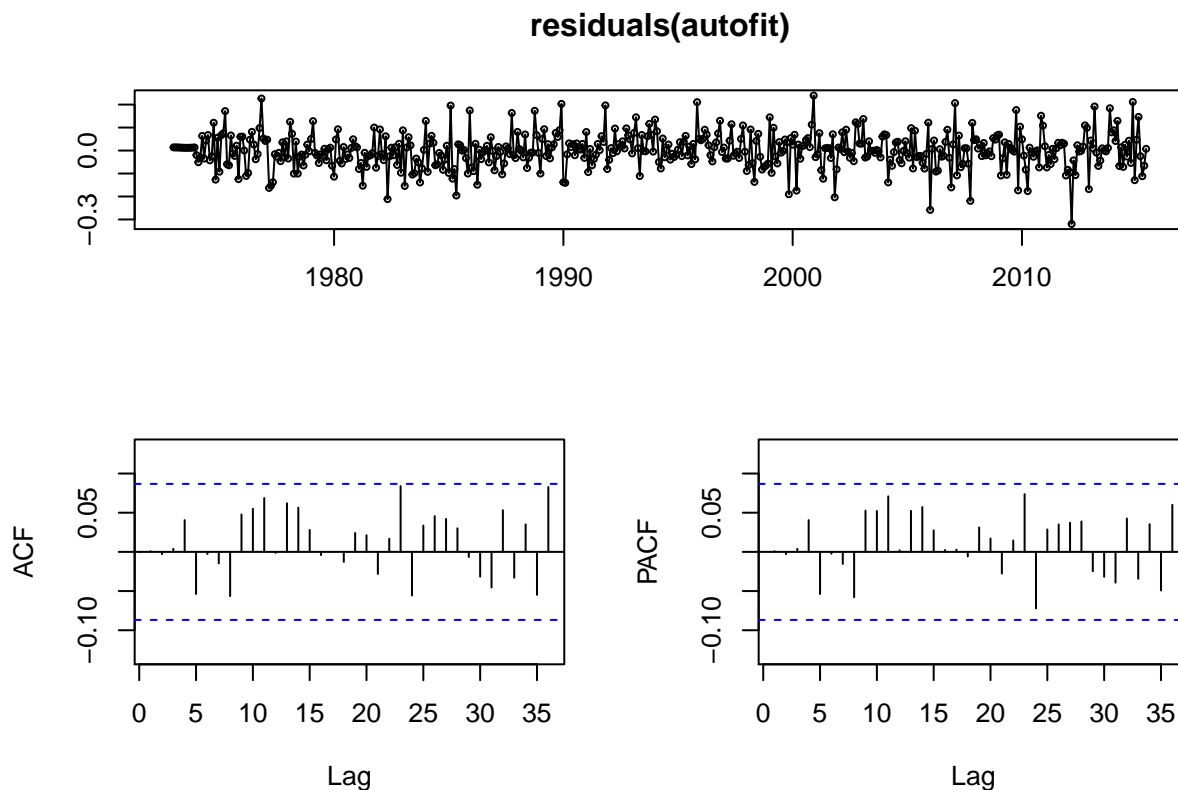
Let us now try a different approach and forecast the log gas consumption with Seasonal ARIMA Models.

Auto Arima on the log data

```
autofit = auto.arima(train, seasonal=TRUE)
print(autofit)
```

```
## Series: train
## ARIMA(2,0,0)(1,1,1)[12] with drift
##
## Coefficients:
##          ar1      ar2      sar1      sma1      drift
##          0.4926  0.0477  0.0011 -0.7531 -3e-04
## s.e.      0.0462  0.0448  0.0600  0.0426  2e-04
##
## sigma^2 estimated as 0.005867:  log likelihood=570.36
## AIC=-1128.71  AICc=-1128.54  BIC=-1103.45
```

```
tsdisplay(residuals(autofit))
```



```
autofk = forecast(autofit,h=12)
accuracy(autofk,test)
```

```
##              ME      RMSE      MAE      MPE      MAPE
## Training set -1.829478e-05 0.07530712 0.05583933 -0.00504198 0.4352824
## Test set     -8.278783e-02 0.12016739 0.08700562 -0.64410685 0.6787249
##              MASE      ACF1 Theil's U
## Training set 0.7213539 0.0009485353    NA
## Test set     1.1239719 0.0958657683 0.2804928
```

The ACF and PACF of the residuals show no significant spikes.

Let us now try with our own seasonal ARIMA. Based on the ACF PACF of the annual differentiation, we try with an ARIMA(4,0,1)(0,1,2).

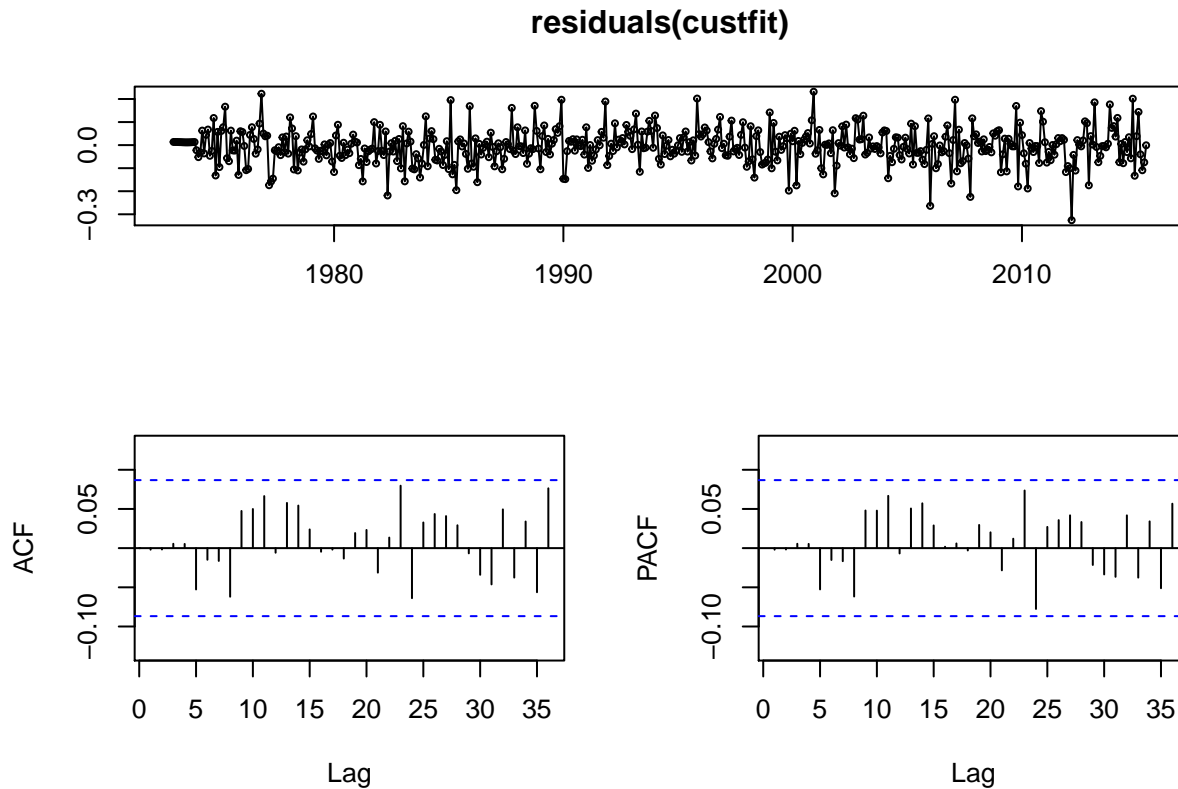
Custom Seasonal Arima on the log data

```
custfit = Arima(train, order=c(4,0,1), seasonal=c(0,1,1))
print(custfit)
```

```
## Series: train
## ARIMA(4,0,1)(0,1,1)[12]
##
## Coefficients:
##      ar1      ar2      ar3      ar4      ma1      sma1
##     -0.0545  0.3166  0.0228  0.0381  0.5493 -0.7441
## s.e.   0.3757  0.1908  0.0499  0.0452  0.3738  0.0337
##
## sigma^2 estimated as 0.005905: log likelihood=569.4
```

```
## AIC=-1124.81 AICc=-1124.58 BIC=-1095.33
```

```
tsdisplay(residuals(custfit))
```



```
custfk = forecast(custfit,h=12)
accuracy(custfk,test)
```

```
##               ME      RMSE      MAE      MPE      MAPE
## Training set -0.005036636 0.07547694 0.05610723 -0.04471939 0.4377556
## Test set     -0.093408622 0.12865998 0.09585320 -0.72825351 0.7483177
##               MASE      ACF1 Theil's U
## Training set 0.7248147 -0.001948896    NA
## Test set     1.2382683 0.114421703 0.3006528
```

Again, we find no alarming spikes in the residuals. But this model is outperformed by the the one found automatically.

Seasonal Arima on the actual data

```
dautofit = auto.arima(dtrain, seasonal=TRUE)
print(dautofit)
```

```
## Series: dtrain
## ARIMA(3,0,3)(0,0,2)[12] with non-zero mean
##
## Coefficients:
##          ar1      ar2      ar3      ma1      ma2      ma3      sma1      sma2
##          0.7523  0.5907 -0.8473  0.0727 -0.7353  0.0931  0.5205  0.2764
## s.e.      0.0534  0.0804  0.0453  0.0837  0.0376  0.0646  0.0491  0.0467
```

```
##          intercept
##          396001.316
## s.e.      4104.313
##
## sigma^2 estimated as 3.764e+09:  log likelihood=-6345.9
## AIC=12711.8   AICc=12712.24   BIC=12754.14
```

```
dautofk = forecast(dautofit,h=12)
accuracy(dautofk,dtest)
```

```
##              ME      RMSE      MAE      MPE      MAPE      MASE
## Training set   323.5301 60805.51 46602.48 -3.560537 15.66163 1.288629
## Test set      -47042.4200 89646.97 70956.37 -29.251244 32.24462 1.962051
##              ACF1 Theil's U
## Training set  0.01116995      NA
## Test set      0.35937203  1.273863
```

This time the RMSE is quite high. Comparing with the results on the log data gives the following RMSE:

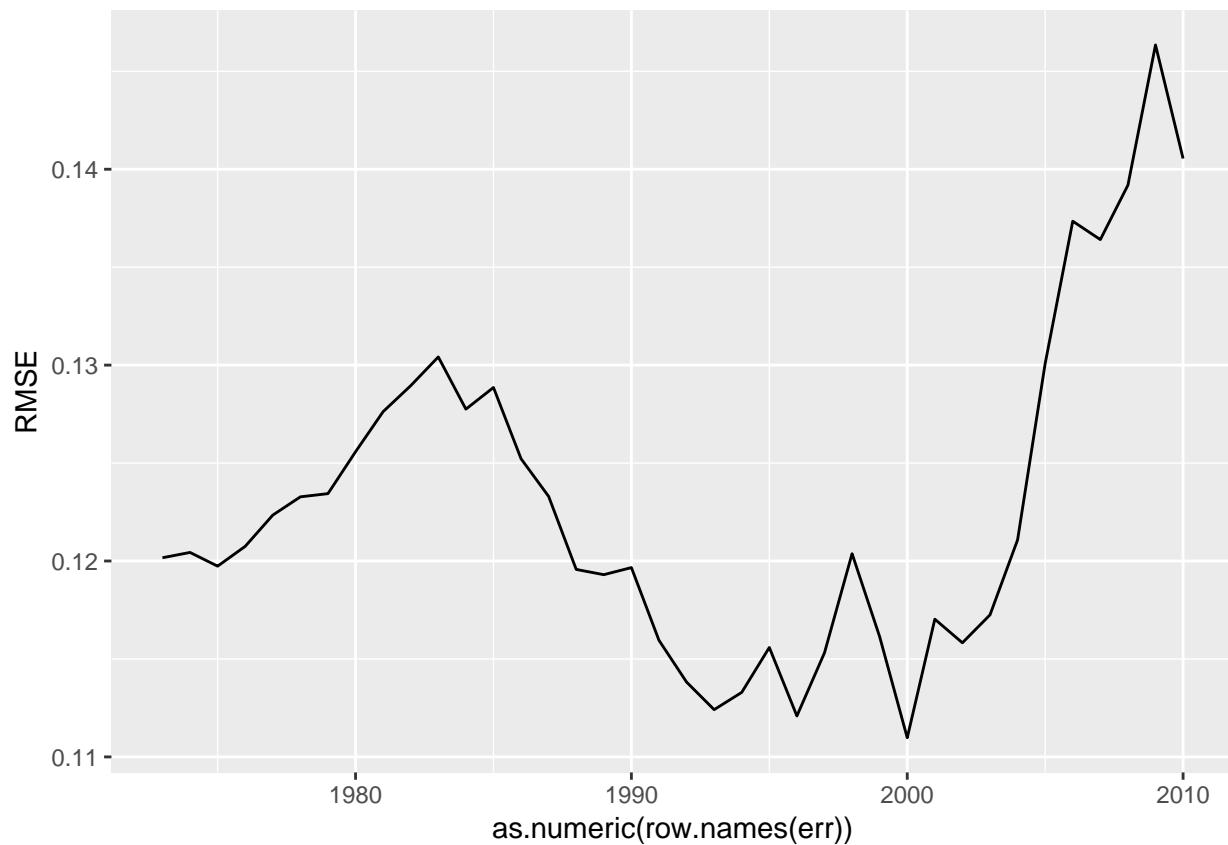
```
kable(data.frame("Direct"=accuracy(dautofk,dtest)[2, "RMSE"],
  "Logarithm"=exp_rmse(dautofk$mean, test), row.names="RMSE"), align='c')
```

	Direct	Logarithm
RMSE	89646.97	67604.67
The sea	sonal ARIMA is not doing any good on the actual data, compared to the log data.	

Rolling Window Evaluation

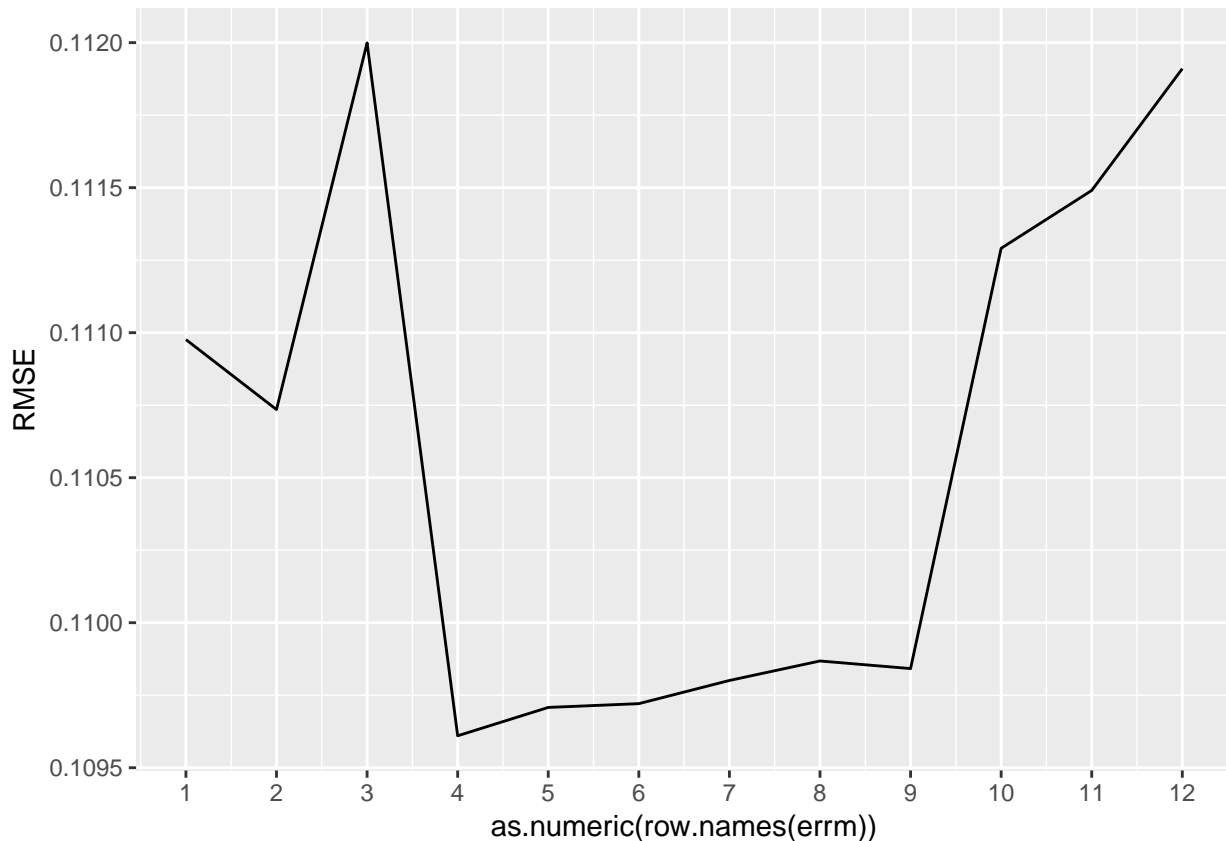
To compare models more precisely, we would like to setup a rolling window evaluation. Let us find the optimal size of that window using the Auto ARIMA we just fitted, by comparing the performances on RMSE of models with different training test time span.

```
err = data.frame("RMSE" = rep(0, 2010-1973+1), row.names = seq(1973, 2010))
for (y in 1973:2010){
  model = Arima(window(train, start=c(y,1)), order=c(2,0,0), seasonal=c(1,1,1), include.drift = TRUE)
  rmse = accuracy(forecast(model, h=12),test)[2,"RMSE"]
  err[as.character(y),] = rmse
}
ggplot(err)+geom_line(aes(x=as.numeric(row.names(err)), y=RMSE))
```

Let us take a closer look at the year 2000.

```
errm = data.frame("RMSE" = rep(0, 12), row.names = seq(1, 12))
for (m in 1:12){
  model = Arima(window(train, start=c(2000,m)), order=c(2,0,0), seasonal=c(1,1,1), include.drift = TRUE)
  rmse = accuracy(forecast(model, h=12),test)[2,"RMSE"]
  errm[as.character(m),] = rmse
}
ggplot(errm)+geom_line(aes(x=as.numeric(row.names(errm)), y=RMSE))+scale_x_continuous(breaks=seq(1,12))
```



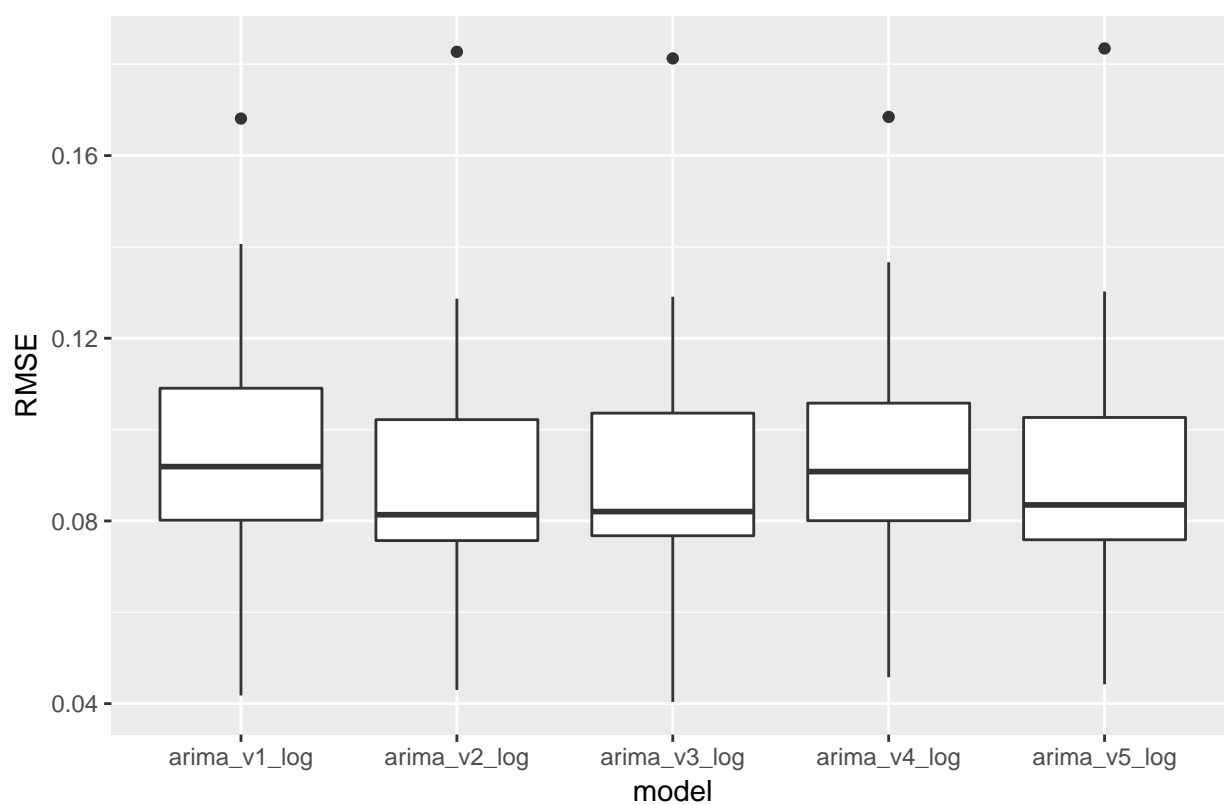
We will thus use a rolling window of 15 years, and define the R function below:

```
roll = function(p,d,q,P,D,Q, drift){
  RMSE = rep(0,28)
  RMSLE = rep(0,28)
  for (y in 1973:2000){
    tr = window(lgas, start=c(y,7), end=c(y+15,6))
    te = window(lgas, start=c(y+15,7), end=c(y+16,6))
    model = Arima(tr, order=c(p,d,q), seasonal=c(P,D,Q), include.drift = drift)
    RMSLE[y+1-1973] = accuracy(forecast(model, h=12),te)[2,"RMSE"]
    RMSE[y+1-1973] = exp_rmse(forecast(model, h=12)$mean, te)
  }
  return(rbind(RMSE, RMSLE))
}
```

We can now, evaluate models predictions on different years. We try below different seasonal arima, reducing the number of lags little by little.

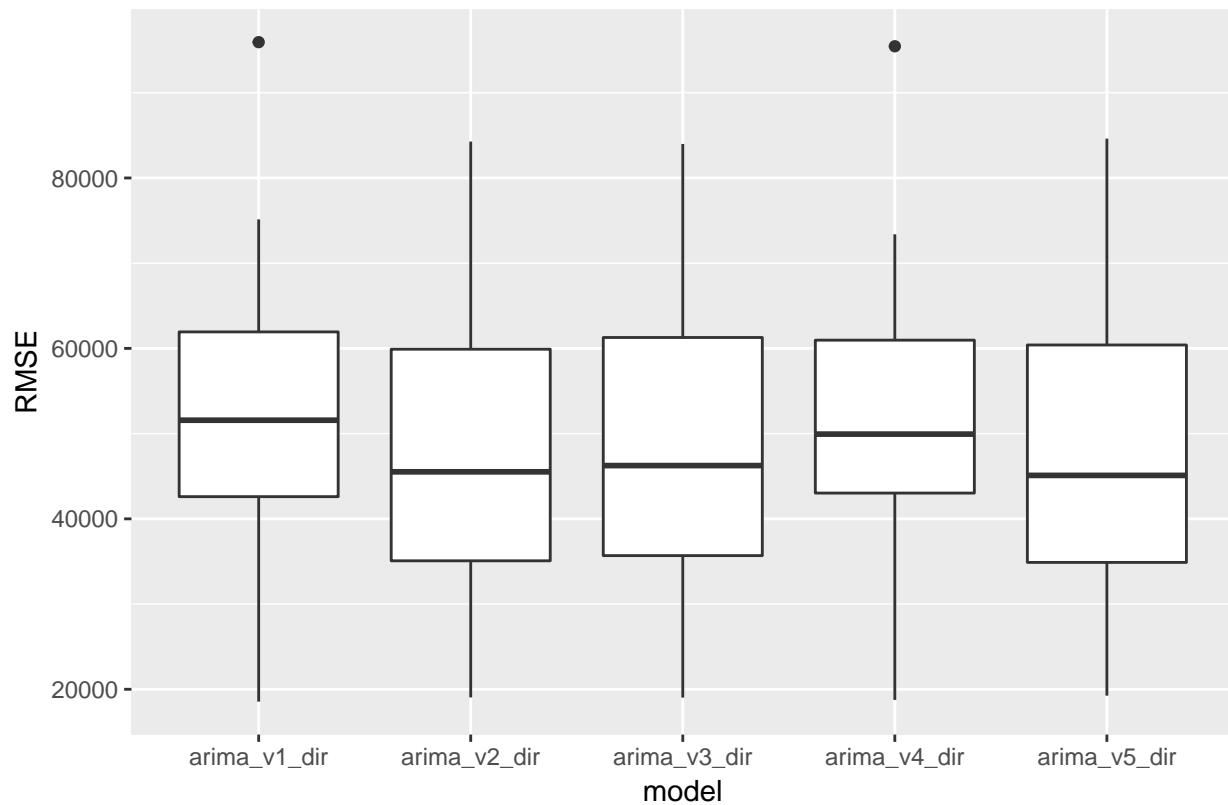
```
eval = data.frame('run'=seq(1:28))
eval[c('arima_v1_dir', 'arima_v1_log')] = t(roll(2,0,0,1,1,1,TRUE))
eval[c('arima_v2_dir', 'arima_v2_log')] = t(roll(4,0,0,0,1,1,FALSE))
eval[c('arima_v3_dir', 'arima_v3_log')] = t(roll(2,0,0,1,1,1,FALSE))
eval[c('arima_v4_dir', 'arima_v4_log')] = t(roll(1,0,0,0,1,1,TRUE))
eval[c('arima_v5_dir', 'arima_v5_log')] = t(roll(1,0,0,0,1,1,FALSE))
ggplot(stack(eval[, (names(eval) != "run") & grepl('log', names(eval))]))+geom_boxplot(aes(x=ind, y=values))
labs(x='model', title="Log Model Performances", y="RMSE")
```

Log Model Performances



```
ggplot(stack(eval[, (names(eval) != "run") & grepl('dir', names(eval))])) + geom_boxplot(aes(x=ind, y=values)) +
  labs(x='model', title="Direct Model Performances", y="RMSE")
```

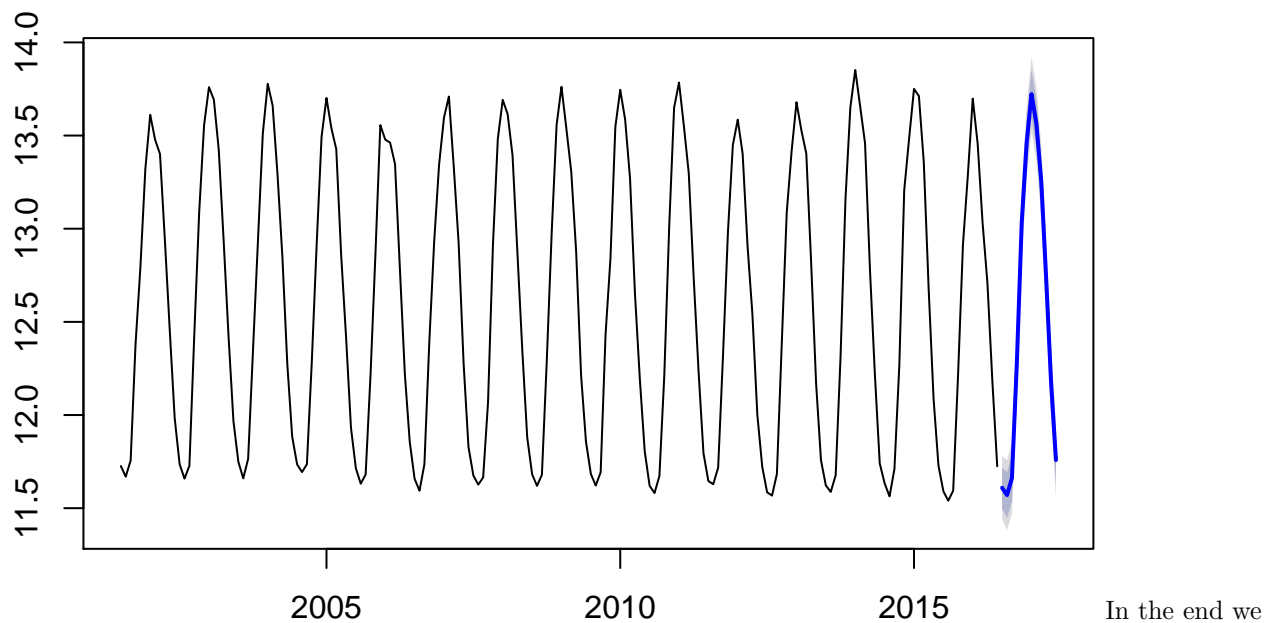
Direct Model Performances



All model have outliers. We keep the model v2, i.e. SARIMA((4,0,0),(0,1,1),12).

```
finalModel = Arima(window(lgas, start=c(2001,7)), order=c(4,0,0), seasonal=c(0,1,1), include.drift = FALSE)
finalFkst = forecast(finalModel, 12)
plot(finalFkst)
```

Forecasts from ARIMA(4,0,0)(0,1,1)[12]



have the following predictions:

```
finalFkst$mean
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 2016                                     11.60963
## 2017 13.72225 13.56325 13.25081 12.72968 12.17312 11.75827
##           Aug      Sep      Oct      Nov      Dec
## 2016 11.57052 11.66087 12.27683 13.02921 13.45850
## 2017
```

```
exp(finalFkst$mean)
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 2016                                     110153.4
## 2017 910957.0 777040.5 568530.5 337620.7 193516.8 127805.8
##           Aug      Sep      Oct      Nov      Dec
## 2016 105928.3 115945.2 214663.3 455528.5 699763.5
## 2017
```

```
write.csv(cbind(exp(finalFkst$mean), finalFkst$mean), "univariate_predictions.csv")
```