ENV 790.30 - Time Series Analysis for Energy Data | Spring 2023

Assignment 8 - Due date 03/27/23

Katherine Burley

Directions

You should open the .rmd file corresponding to this assignment on RStudio. The file is available on our class repository on Github. And to do so you will need to fork our repository and link it to your RStudio.

Once you have the project open the first thing you will do is change "Student Name" on line 3 with your name. Then you will start working through the assignment by **creating code and output** that answer each question. Be sure to use this assignment document. Your report should contain the answer to each question and any plots/tables you obtained (when applicable).

When you have completed the assignment, **Knit** the text and code into a single PDF file. Rename the pdf file such that it includes your first and last name (e.g., "LuanaLima_TSA_A08_Sp22.Rmd"). Submit this pdf using Sakai.

Set up

Some packages needed for this assignment: `forecast`,`tseries`,`smooth`. Do not forget to load them before running your script, since they are NOT default packages.

```
#Load/install required package here
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(tseries)
library(smooth)
```

```
## Warning: package 'smooth' was built under R version 4.2.3
```

```
## Loading required package: greybox
```

```
## Warning: package 'greybox' was built under R version 4.2.3
```

```
## Package "greybox", v1.0.7 loaded.
```

```
## This is package "smooth", v3.2.0
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2
## --
```

```
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::spread() masks greybox::spread()
```

```
library(ggplot2)
library(readxl)
library(Kendall)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
##
## The following object is masked from 'package:greybox':
##
##     hm
##
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
# Set option for text to wrap in PDF output
knitr::opts_chunk$set(tidy.opts = list(width.cutoff = 60), tidy = TRUE)
```

Importing and processing the data set

Consider the data from the file "inflowtimeseries.txt". The data corresponds to the monthly inflow in m^3/s for some hydro power plants in Brazil. You will only use the last column of the data set which represents one hydro plant in the Amazon river basin. The data span the period from January 1931 to August 2011 and is provided by the Brazilian ISO.

For all parts of the assignment prepare the data set such that the model consider only the data from January 2000 up to December 2009. Leave the year 2010 of data (January 2010 to December 2010) for the out-of-sample analysis. Do **NOT** use data fro 2010 and 2011 for model fitting. You will only use it to compute forecast accuracy of your model.

Part I: Preparing the data sets

Q1

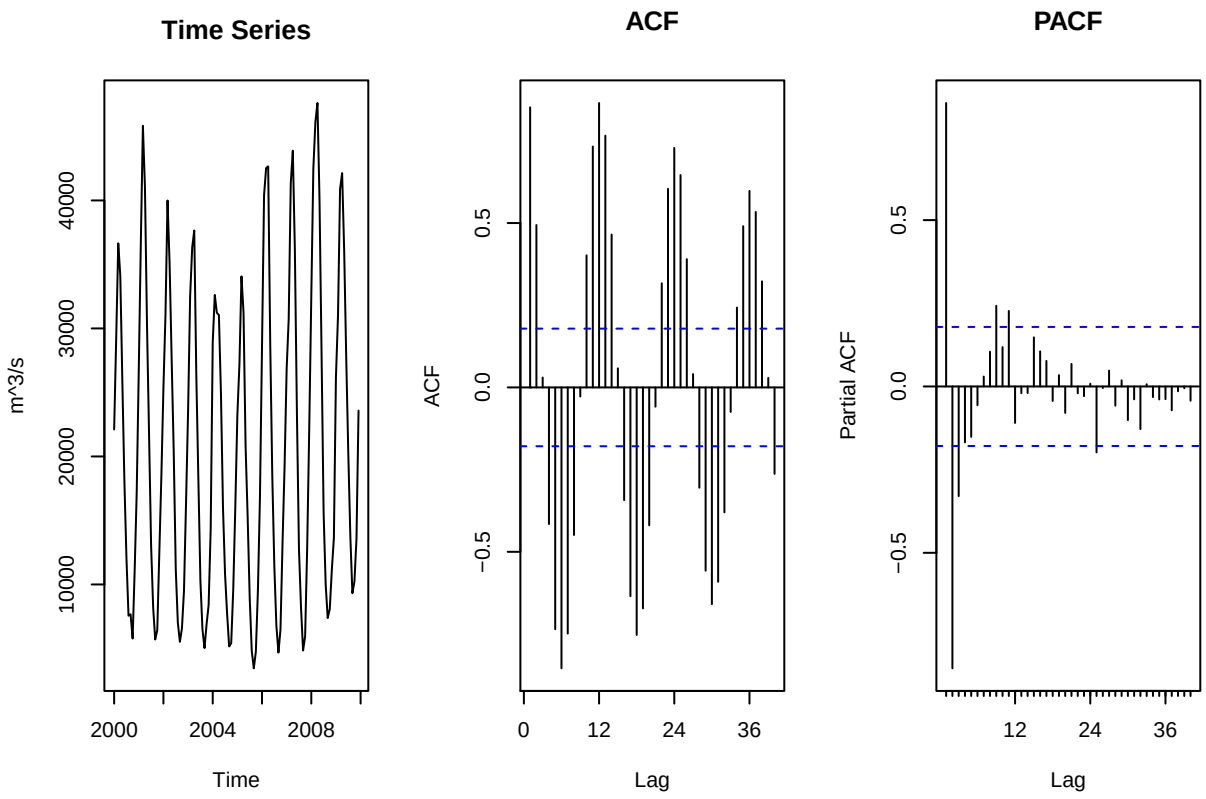
Read the file into a data frame. Prepare your time series data vector such that observations start in January 2000 and end in December 2009. Make you sure you specify the **start=** and **frequency=** arguments. Plot the time series over time, ACF and PACF.

```
# Read the file into a dataframe
inflow_data <- read_table("../Data/inflowtimeseries.txt", col_names = FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_character(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
```

```
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double(),
##   X11 = col_double(),
##   X12 = col_double(),
##   X13 = col_double(),
##   X14 = col_double(),
##   X15 = col_double(),
##   X16 = col_double(),
##   X17 = col_double(),
##   X18 = col_logical()
## )
```

```r
inflow_data <- inflow_data %>%
    select(X1, X2, X17) %>%
    rename(month = X1, year = X2, inflow = X17) %>%
    mutate(month_year = paste(month, year, sep = "-")) %>%
    mutate(date = my(month_year)) %>%
    filter(year %in% seq(2000, 2010))

# Prepare time series vector
inflow_data_train <- inflow_data %>%
    filter(year %in% seq(2000, 2009))

inflow_data_test <- inflow_data %>%
    filter(year == 2010)

inflow_train_ts <- ts(inflow_data_train$inflow, start = 2000,
    freq = 12)
inflow_test_ts <- ts(inflow_data_test$inflow, start = 2010, freq = 12)

inflow_ts <- ts(inflow_data$inflow, start = 2000, freq = 12)
# also create ts of whole series to deseason later

# Plot the time series over time, ACF and PACF
par(mfrow = c(1, 3), mar = c(6, 4, 4, 2))
plot(inflow_train_ts, main = "Time Series", ylab = "m^3/s")
Acf(inflow_train_ts, lag.max = 40, main = "ACF")
Pacf(inflow_train_ts, lag.max = 40, main = "PACF")
```

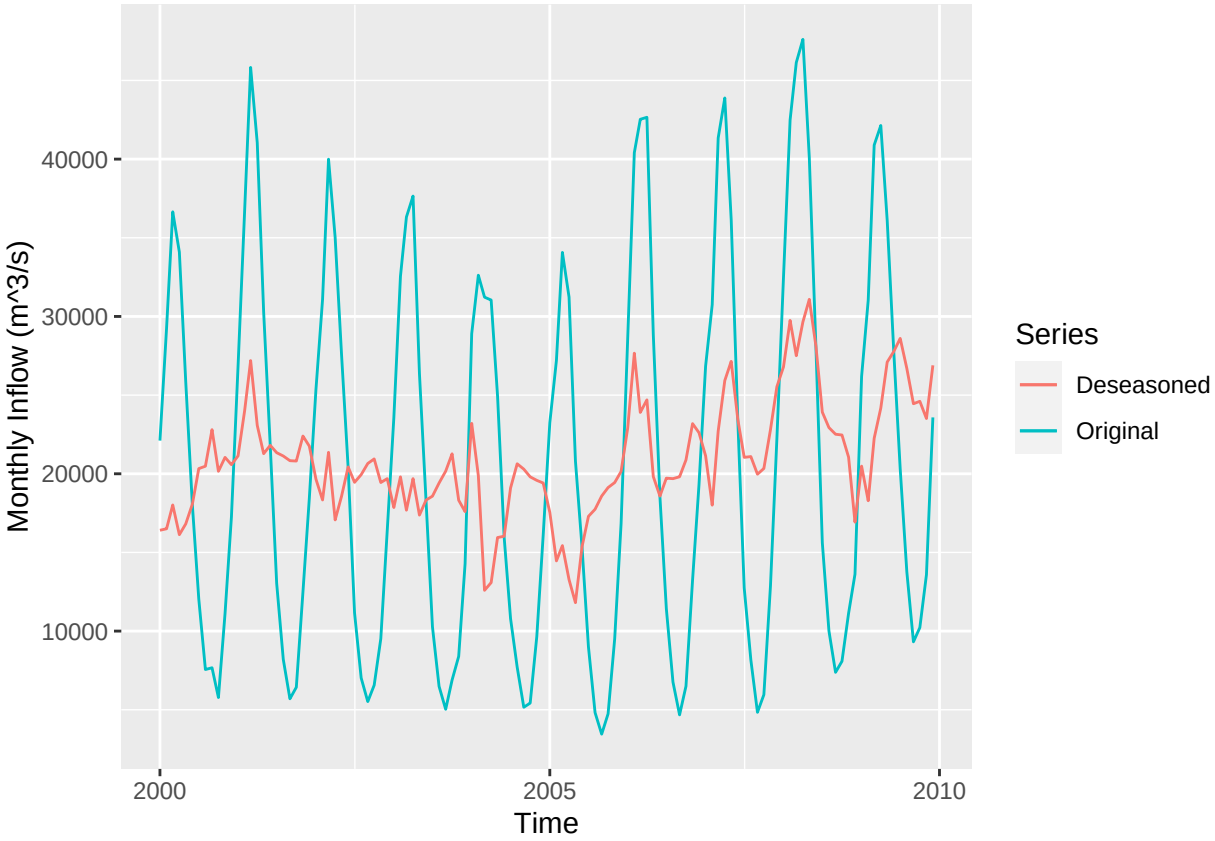


Q2

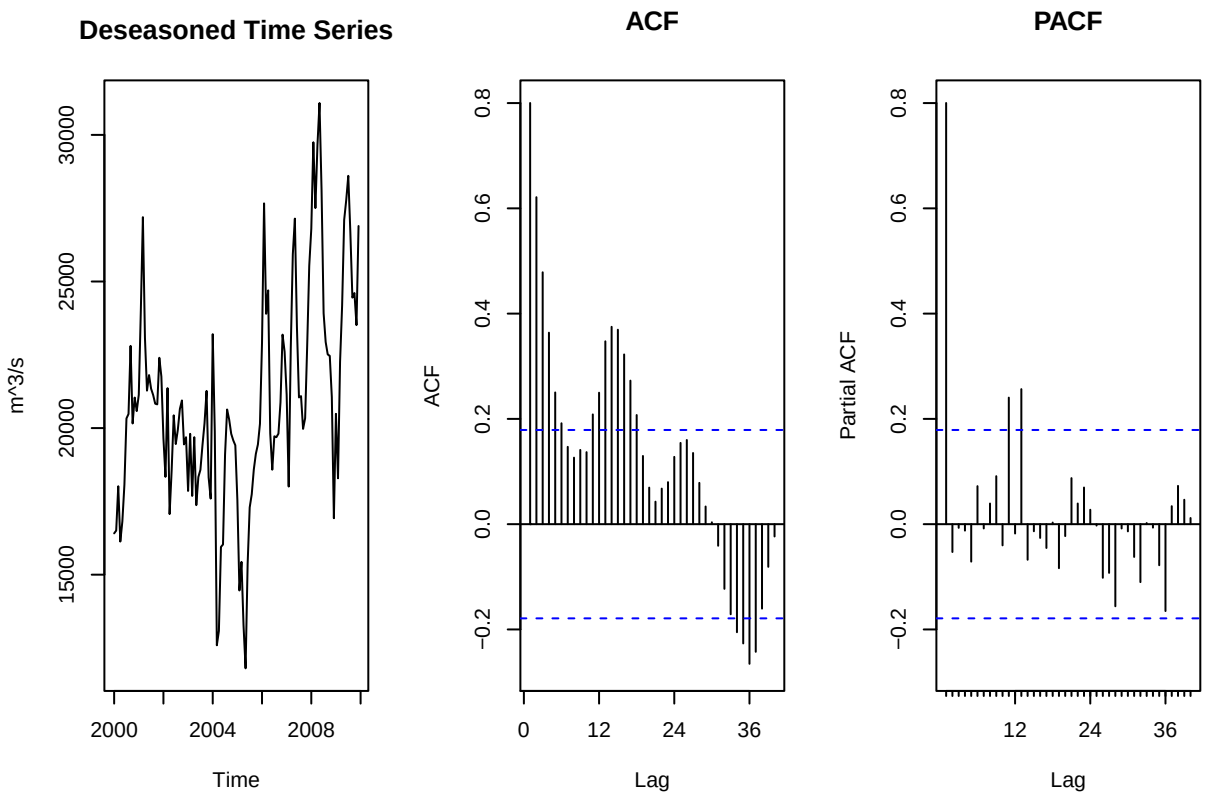
Using the *decompose*() or *stl*() and the *seasadj*() functions create a series without the seasonal component, i.e., a deseasonalized inflow series. Plot the deseasonalized series and original series together using ggplot, make sure your plot includes a legend. Plot ACF and PACF for the deaseasonalized series. Compare with the plots obtained in Q1.

```
# Create deseasonalized data series
decompose_inflow <- decompose(inflow_ts, type = "additive")
deseasonal_inflow <- seasadj(decompose_inflow)

deseas_inflow_train <- ts(deseasonal_inflow[1:120], start = 2000,
    freq = 12)
deseas_inflow_test <- ts(deseasonal_inflow[121:132], start = 2010,
    freq = 12)

# Plot the deseasonalized series and original series
# together using ggplot Make sure your plot includes a
# legend
ggplot() + geom_line(aes(x = inflow_data_train$date, y = inflow_train_ts,
    col = "Original")) + geom_line(aes(x = inflow_data_train$date,
    y = deseas_inflow_train, col = "Deseasoned")) + xlab("Time") +
    ylab("Monthly Inflow (m^3/s)") + labs(color = "Series")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```




```
# Plot ACF and PACF for the deaseasonalized series.
# Compare with the plots obtained in Q1.
par(mfrow = c(1, 3), mar = c(6, 4, 4, 2))
plot(deseas_inflow_train, main = "Deseasoned Time Series", ylab = "m^3/s")
Acf(deseas_inflow_train, lag.max = 40, main = "ACF")
Pacf(deseas_inflow_train, lag.max = 40, main = "PACF")
```



In the original series, the ACF had a distinct seasonal patterns, peaking about every six lags (months). In the deseasoned series, the seasonal pattern in the ACF is reduced substantially. The series appears to decay, although there is still some cyclicality, with peaks occuring at 12 and 36 lags. The PACF plots are similar, except that the PACF for the deseasoned series has less lags with significant values and does not have a high, negative PACF value at lag 2 like the original series does.

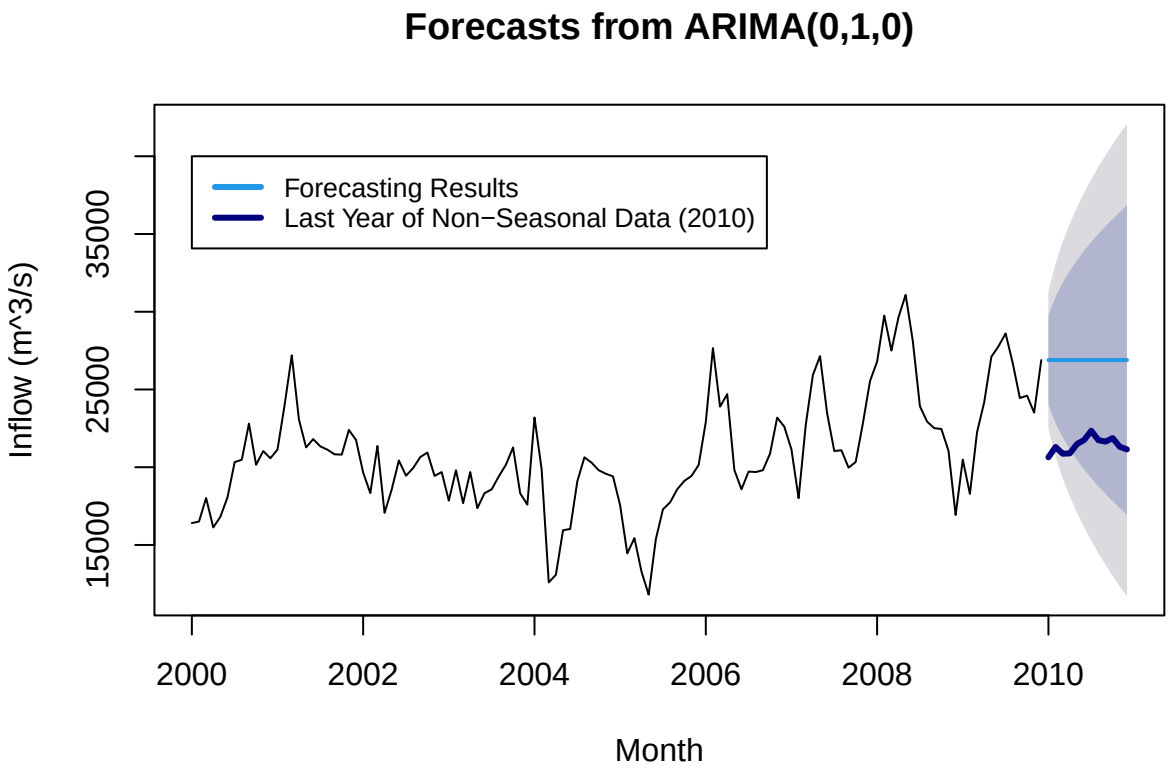
Part II: Forecasting with ARIMA models and its variations

Q3

Fit a non-seasonal ARIMA(p, d, q) model using the auto.arima() function to the non-seasonal data. Forecast 12 months ahead of time using the $forecast()$ function. Plot your forecasting results and further include on the plot the last year of non-seasonal data to compare with forecasted values (similar to the plot on the lesson file for M10).

```
# Fit non-seasonal ARIMA model to non-seasonal data
ARIMA_autofit <- auto.arima(deseas_inflow_train, max.D = 0, max.P = 0,
    max.Q = 0)  # non-seasonal ARIMA

# Forecast 12 months ahead of time
ARIMA_forecast <- forecast(object = ARIMA_autofit, h = 12)
# plot(ARIMA_forecast)

# Plot forecasting results and include last year of
# non-seasonal data to compare with forecasted values Note:
# Interpreting this as comparing the 12 month ahead
# forecast (2010) to the deseasoned, held-out 2010 values
plot(ARIMA_forecast, xlab = "Month", ylab = "Inflow (m^3/s)")
```

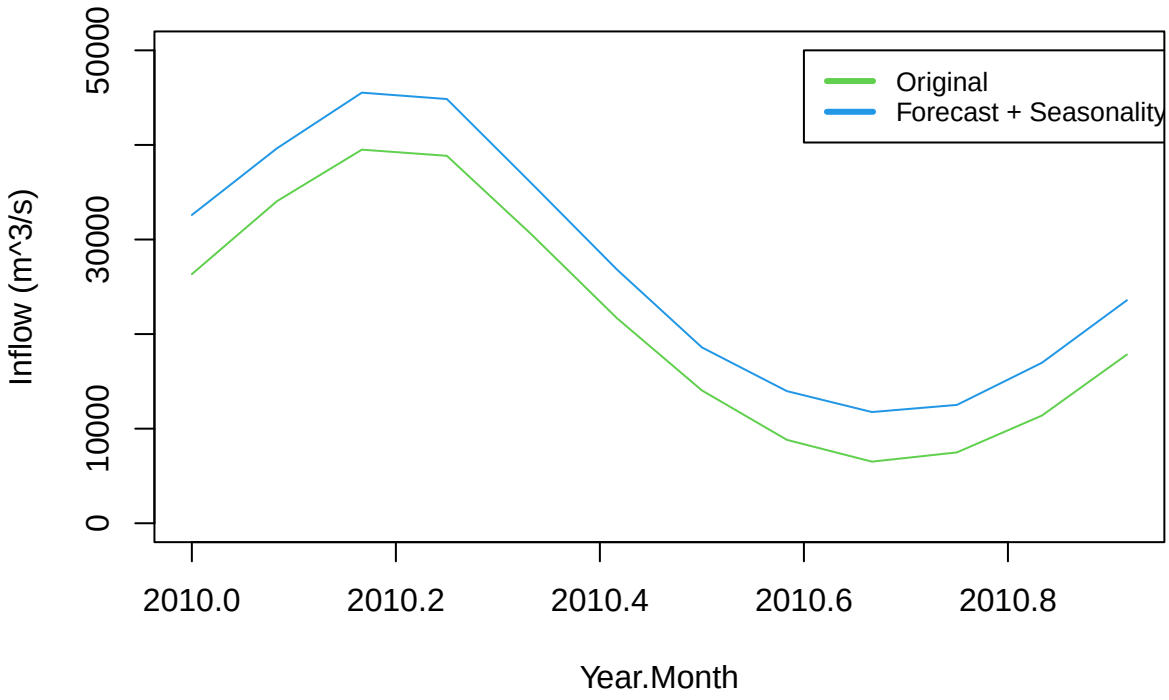
```
lines(deseas_inflow_test, type = "l", col = "navy", lwd = 3)
legend(2000, 40000, legend = c("Forecasting Results", "Last Year of Non-Seasonal Data (2010)"),
    col = c(4, "navy"), lwd = 3, cex = 0.8)
```



Q4

Put the seasonality back on your forecasted values and compare with the original seasonal data values. *Hint* : One way to do it is by summing the last year of the seasonal component from your decompose object to the forecasted series.

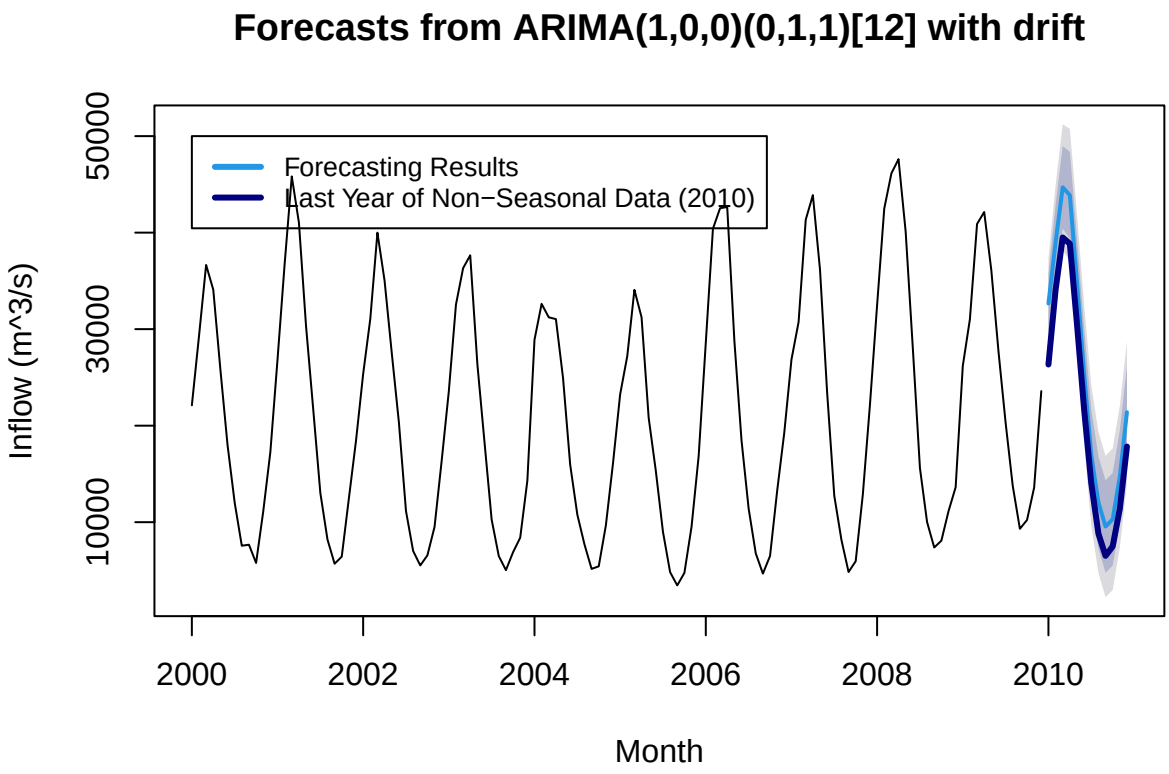
```
# Put the seasonality back on your forecasted values and
# compare with the original seasonal data values.
# Interpreting this as comparing forecasted values +
# seasonality for 2010 to the original 2010 data that was
# held out
seasonal_comp_forecast <- decompose_inflow$seasonal[121:132]
ARIMA_forecast_plusseas <- ARIMA_forecast$mean + seasonal_comp_forecast

# compare with the original seasonal data values
plot(inflow_test_ts, xlab = "Year.Month", ylab = "Inflow (m^3/s)",
    ylim = c(0, 50000), col = 3)
lines(ARIMA_forecast_plusseas, col = 4)
legend(2010.6, 50000, legend = c("Original", "Forecast + Seasonality"),
    col = c(3, 4), lwd = 3, cex = 0.8)
```



The forecast plus seasonality component is very similar to the original series, but higher by roughly around 6000 m^3/s at each month. ### Q5

Repeat Q3 for the original data, but now fit a seasonal ARIMA$(p,d,q)x(P,D,Q)_{12}$ also using the auto.arima().

```
# Repeat Q3 for the original data, but now fit a seasonal
# ARIMA$(p,d,q)x(P,D,Q)_ {12}$ also using the auto.arima()

# Fit seasonal ARIMA model to original data
SARIMA_autofit <- auto.arima(inflow_train_ts, seasonal = TRUE)  # allows P,D,Q to vary

# Forecast 12 months ahead of time
SARIMA_forecast <- forecast(object = SARIMA_autofit, h = 12)
# plot(SARIMA_forecast)

# Plot forecasting results and include last year of
# non-seasonal data to compare with forecasted values Note:
# Interpreting this as comparing the 12 month ahead
# forecast (2010) to the deseasoned, held-out 2010 values
plot(SARIMA_forecast, xlab = "Month", ylab = "Inflow (m^3/s)")
lines(inflow_test_ts, type = "l", col = "navy", lwd = 3)
legend(2000, 50000, legend = c("Forecasting Results", "Last Year of Non-Seasonal Data (2010)"),
    col = c(4, "navy"), lwd = 3, cex = 0.8)
```

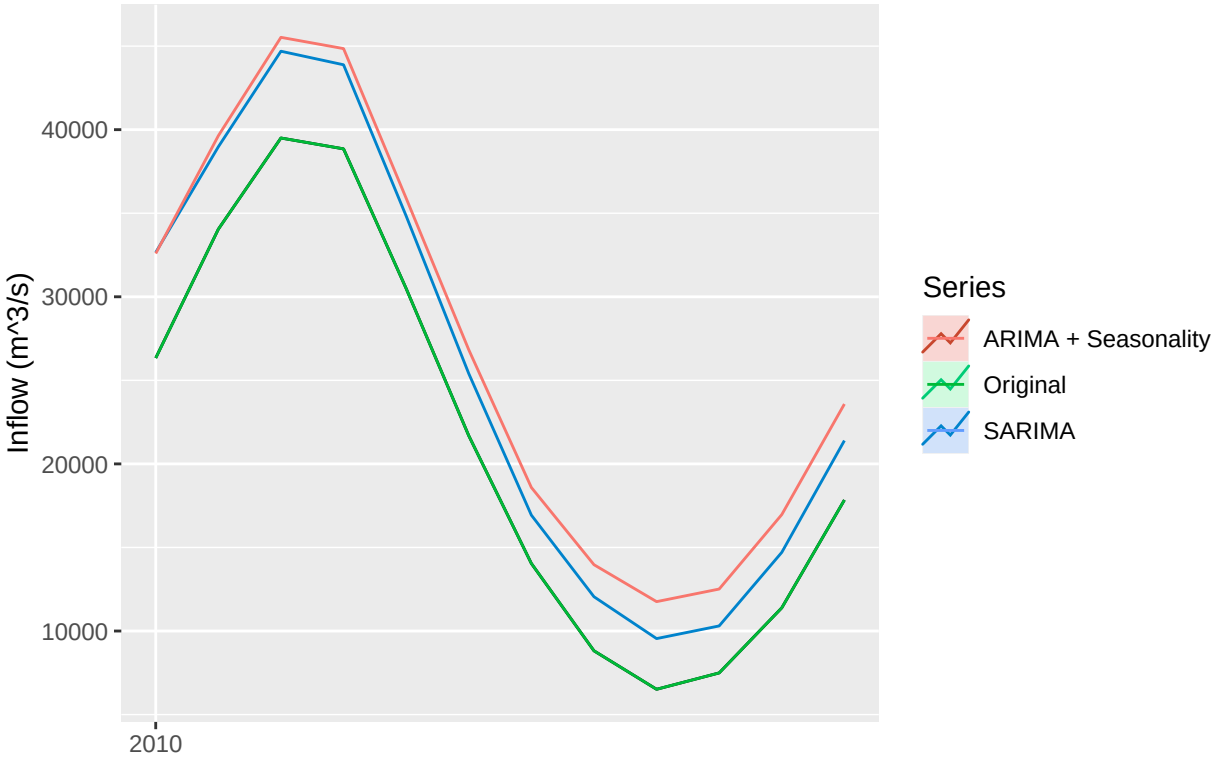


Q6

Compare the plots from Q4 and Q5 using the autoplot() function.

```
autoplot(inflow_test_ts) + autolayer(inflow_test_ts, series = "Original",
    PI = FALSE) + autolayer(SARIMA_forecast, series = "SARIMA",
    PI = FALSE) + autolayer(ARIMA_forecast_plusseas, series = "ARIMA + Seasonality",
    PI = FALSE) + ylab("Inflow (m^3/s)") + xlab("") + labs(col = "Series")
```

```
## Warning in ggplot2::geom_line(ggplot2::aes_(x = ~timeVal, y = ~seriesVal, : Ignoring unknown paramet
## Ignoring unknown parameters: `PI`
```



The SARIMA forecasting results are slightly closer to the original series than ARIMA + seasonality component.

Part III: Forecasting with Other Models

Q7

Fit an exponential smooth model to the original time series using the function *ses*() from package `forecast`. Note that this function automatically do the forecast. Do not forget to set the arguments: silent=FALSE and holdout=FALSE, so that the plot is produced and the forecast is for the year of 2010.

```
# Fit an exponential smooth model to the original time
# series using ses()
SES_results = ses(y = inflow_train_ts, h = 12, holdout = FALSE,
    silent = FALSE)
```

Part IV: Checking Forecast Accuracy

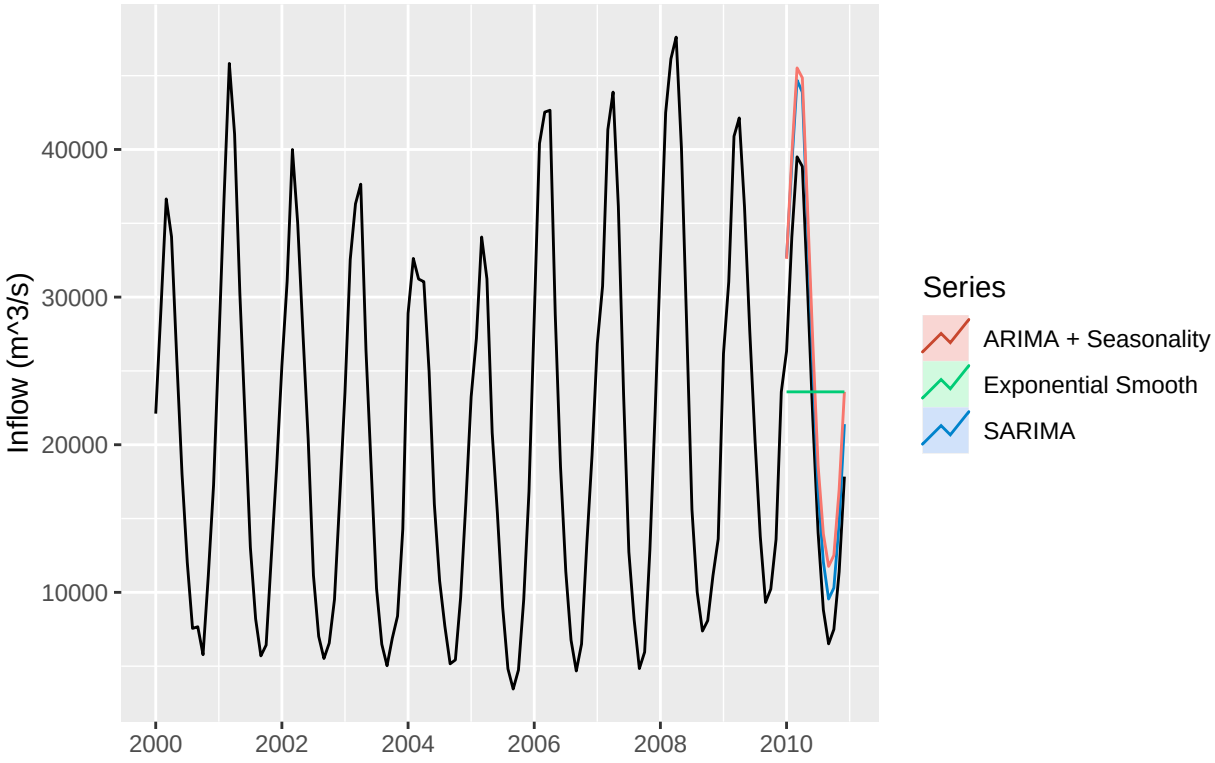
Q8

Make one plot with the complete original seasonal historical data (Jan 2000 to Dec 2010). Now add the forecasts from each of the developed models in parts Q4, Q5, Q7 and Q8. You can do it using the autoplot() combined with autolayer(). If everything is correct in terms of time line, the forecasted lines should appear only in the final year. If you decide to use ggplot() you will need to create a data frame with all the series will need to plot. Remember to use a different color for each model and add a legend in the end to tell which forecast lines corresponds to each model.

```
# Make one plot with the complete original seasonal
# historical data
```

```
autoplot(inflow_ts) + autolayer(SARIMA_forecast, series = "SARIMA",
    PI = FALSE) + autolayer(ARIMA_forecast_plusseas, series = "ARIMA + Seasonality",
    PI = FALSE) + autolayer(SES_results, series = "Exponential Smooth",
    PI = FALSE) + ylab("Inflow (m^3/s)") + xlab("") + labs(col = "Series")
```

```
## Warning in ggplot2::geom_line(ggplot2::aes_(x = ~timeVal, y = ~seriesVal, :
## Ignoring unknown parameters: `PI`
```



Q9

From the plot in Q9 which model or model(s) are leading to the better forecasts? Explain your answer. Hint: Think about which models are doing a better job forecasting the high and low inflow months for example.

> Answer: The SARIMA model appears to be producing the best forecast in this case. The exponential smoothing model only yields a straight line and does not follow the seasonal trend at all. The ARIMA plus seasonality and SARIMA are similar and in both cases they project a higher value than the observed seasonal high inflow month (over-project) and a higher value than the observed low inflow month (under-project) during the test year, but the SARIMA is closer to the observed values throughout the test period. This can be seen even more clearly in the plot in Q6.

Q10

Now compute the following forecast metrics we learned in class: RMSE and MAPE, for all the models you plotted in part Q9. You can do this by hand since your have forecasted and observed values for the year of 2010. Or you can use R function *accuracy*() from package "forecast" to do it. Build and a table with the results and highlight the model with the lowest MAPE. Does the lowest MAPE corresponds match your answer for part Q10?

```r
# Now compute the following forecast metrics we learned in
# class: RMSE and MAPE, build and a table with the results
# and highlight the model with the lowest MAPE
ARIMA_scores <- accuracy(ARIMA_forecast_plusseas, inflow_test_ts)
ARIMA_scores <- as.data.frame(ARIMA_scores)
ARIMA_scores <- ARIMA_scores %>%
    mutate(model = "ARIMA")

SARIMA_scores <- accuracy(SARIMA_forecast$mean, inflow_test_ts)
SARIMA_scores <- as.data.frame(SARIMA_scores)
SARIMA_scores <- SARIMA_scores %>%
    mutate(model = "SARIMA")

SES_scores <- accuracy(SES_results$mean, inflow_test_ts)
SES_scores <- as.data.frame(SES_scores)
SES_scores <- SES_scores %>%
    mutate(model = "SES")

compare <- as.data.frame(rbind(ARIMA_scores, SARIMA_scores, SES_scores))

best_model_index <- which.min(compare[, "MAPE"])
cat("The best model by MAPE is:", compare[best_model_index, ]$model)
```

```
## The best model by MAPE is: SARIMA
```

Does the lowest MAPE corresponds match your answer for part Q10? > Answer: Yes, the SARIMA model that appeared to be the best fit based on the plot in Q9 also has the lowest MAPE! And the lowest RMSE.