DSO-522 FINAL PROJECT

with the love of automation

Abstract

This is a project devoted to discover a way to automate the manual process discussed in DSO-522 class, where students are asked to perform ARIMA analysis and give insights to audience.

Wu, Shu

swu173@usc.edu

Table of Contents

Executive Summary

This project, in general, is about an automated version of the time-series analysis process for ARIMA modeling, which Professor Gabrys has been sharing with us in the DSO-522 classes. To make business students, especially MBA students feel more comfortable in using R to do data analysis, this project article will serve as a “manual” for the automated function, and in the latter part, take in a real-life data analysis as an example.

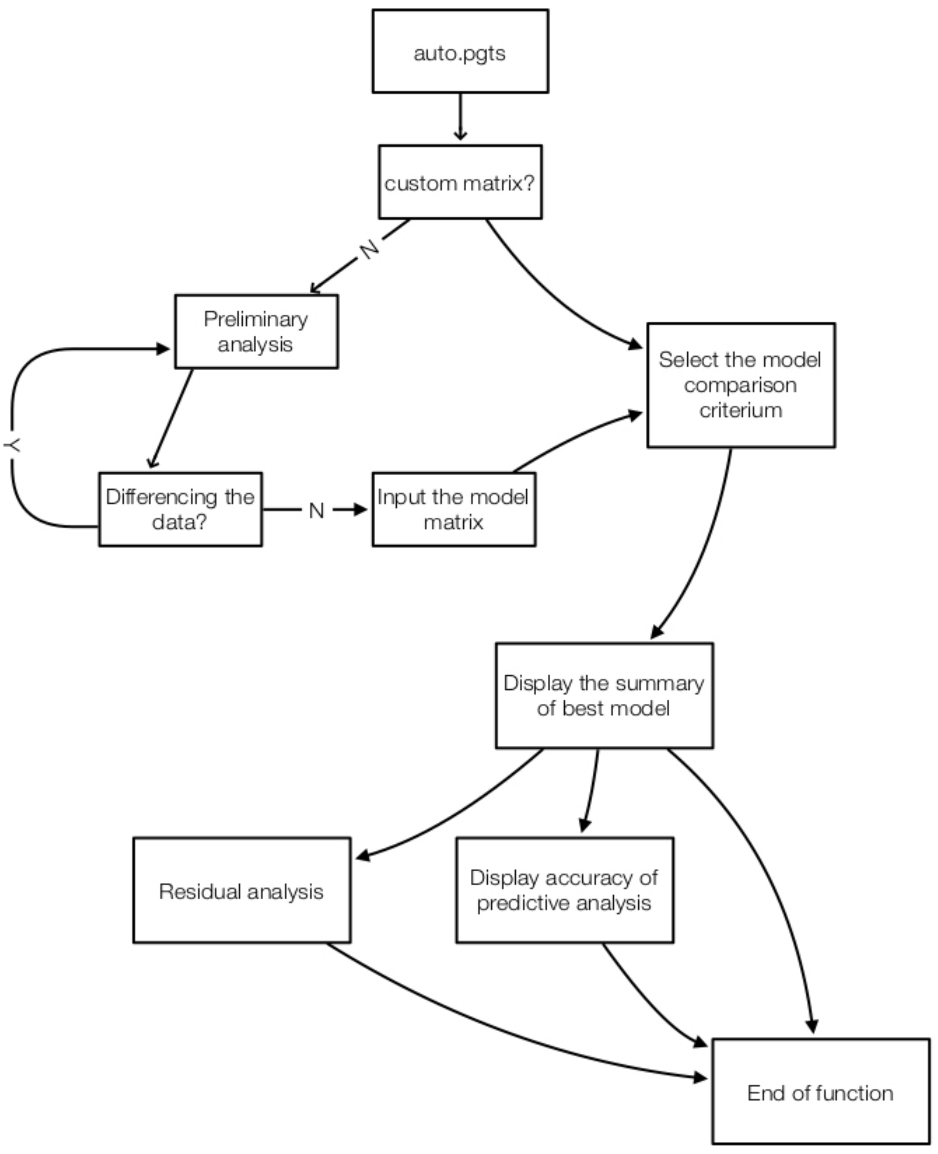
Introduction

ARIMA models play a huge part in time-series data analysis, with its versatility in accurately modeling the movement of volatile data. However, for people who have no background in statistics or econometrics, they will tend to find it difficult and baffling to follow the data analyzing process. Actually, after reading through professor’s in-class notes and code, I start to believe that there’s should be a more user-friendly way and simple way for people who doesn’t like to code a lot just to see a little result. Inspired by this fact, I set off myself to find a less technical way to go through the ARIMA modeling procedure.

Methodology

Visualization of how the function works

To start with the description of the function, I’d like to show a flow chart that can make the illustration easier. In general, this function will, firstly, show you the general idea of how the data looks like, and if you’d like to difference the data for the sake of stationarity. Secondly, it will ask you to input the order number that you consider useful in the ARIMA model, which will be stored and used to determine the best possible model. Thirdly, it will display the graph and summary for the best model for your analysis. Last but not least, you will be provided two options whether or not to perform the residual analysis and the predictive analysis. Overall, this function has sort out the analyzing procedure used in class for the manual process to find out a workable ARIMA model.



Detailed explanations for the terminology

The name of the function is called as "auto.pgts", which is in short of "Professor Gabrys' Time-series Analysis", and is made to be easily remembered. This function takes in three parameters: "data", "residual" and "pred".

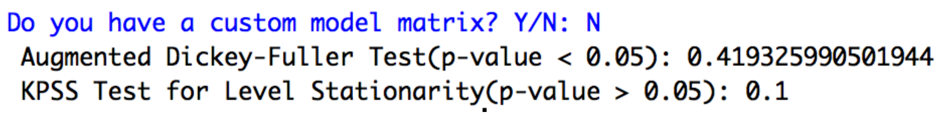
- "data" is the time-series data for the preliminary analysis.
- "residual" takes in a logical value specifying whether or not to do the residual analysis.
- "pred" also takes in a logical value specifying whether or not to do the predictive analysis.

This function, in fact, requires more than those three values to be functional, but to keep it simple in form, I decide to change the interaction between the user and this function from the parameter bracket to the console. This function will constantly ask the user to input some simple values in the console, during the running period. By doing this, I believe it is better for non-technical people to be able to understand and use the function.

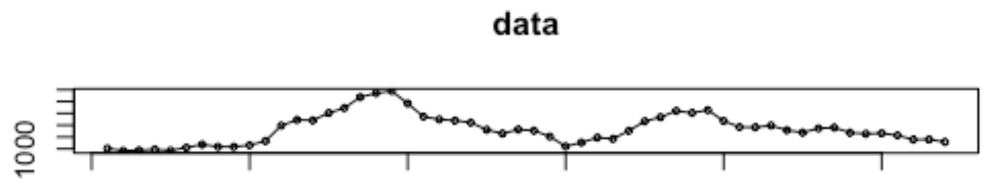
Custom Matrix

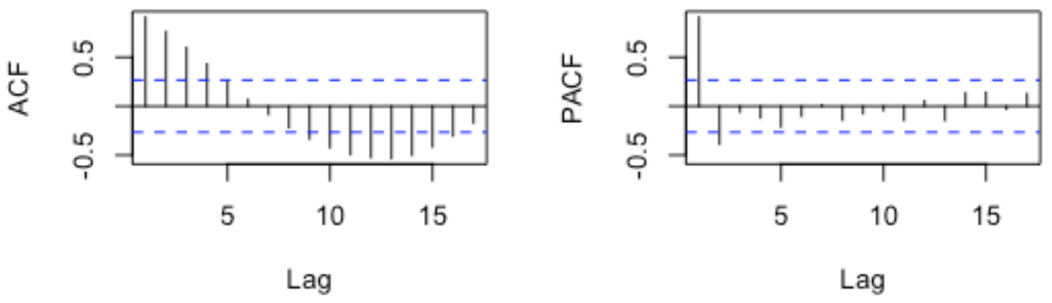
At the beginning of the function, it will ask the user if you'd like to use your own model matrix. A model matrix here refers to an n * 6 matrix in which each row contains the 6 numbers that will be used as the order in ARIMA function. Specifically, the 1st to 3rd represent the "order = c(a, b, c)" option, and the 4th to 6th represent the "seasonal = c(a, b, c)" option in the ARIMA function. Therefore, if the user has done the preliminary analysis elsewhere other than in the auto.pgts() function, he will have a chance to jump over the analysis part and go directly to the model-comparing part of this function.

Preliminary analysis



```
Do you have a custom model matrix? Y/N: N
 Augmented Dickey-Fuller Test(p-value < 0.05): 0.419325990501944
 KPSS Test for Level Stationarity(p-value > 0.05): 0.1
```





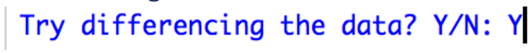
For users who don't have the custom model matrix to start ARIMA modeling right at once, this function will provide the time-series graph as well as other useful statistics for them to help determine the proper orders in the ARIMA function.

- Graph for time-series data

- ACF and PACF graphs
- Augmented Dickey-Fuller Test
- KPSS Test for Level Stationarity

With the common tools above that we've used throughout the course, I am hoping that user can then determine if the data is stationary, and if it is, what are the potential models?

Differencing the data



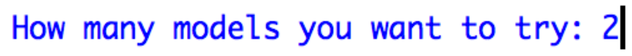
```
Try differencing the data? Y/N: Y
```

This is an optional sector for users. After the preliminary analysis section, the user will be provided with an option whether or not to difference the data, and redo the preliminary analysis until he/she feels the graph is good enough for further analysis. In sum, this section will be a loop with the same question pops up every time at the end of the preliminary analysis, until the user inputs "N" to settle on that dataset.

Specifically, here, user needs to input the differenced data in the diff() function, with data name being "data", and for the remaining part of this function, it will uses the data that is specified here. If the user decide not to difference the data, when the initial data is good enough, the original data will be carried over.

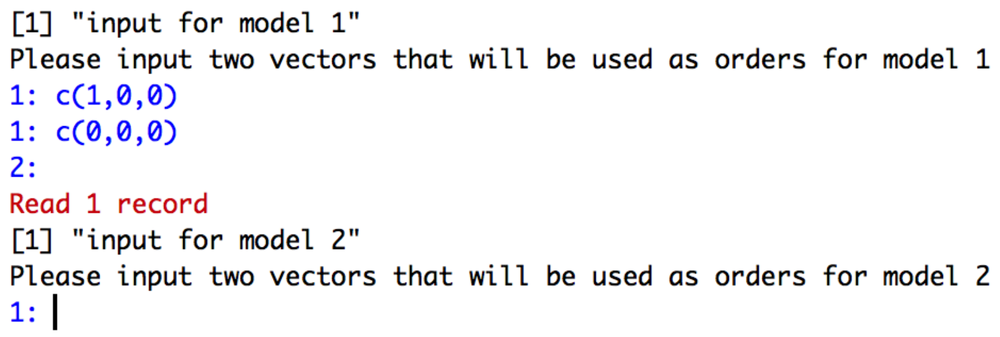
Create the model matrix

Whenever the user thinks the data is analysis-ready, he/she can start to create the model matrix, which in this function will be very user-friendly. To start off, the user will be asked about the number of models he wants to discover.



```
How many models you want to try: 2
```

The number being input here, will be then used as the *n* in the *n*6* matrix as we described in the custom matrix part. Secondly, after deciding on how many models to be discovered, the function will then ask about the order to be used in each model, just as we discussed in class. For example, for the first model, I'd like to use "order = c(1,0,0)" and no seasonal order, I can input the values as the following way:

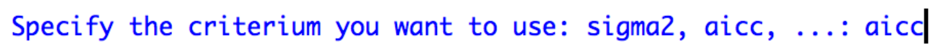


```
[1] "input for model 1"
Please input two vectors that will be used as orders for model 1
1: c(1,0,0)
1: c(0,0,0)
2:
Read 1 record
[1] "input for model 2"
Please input two vectors that will be used as orders for model 2
1: 
```

In the function, for these types of input, I use the scan() function, so please remember to hit the "Enter" button when you have done for any model, and then the function will jump over to the next input.

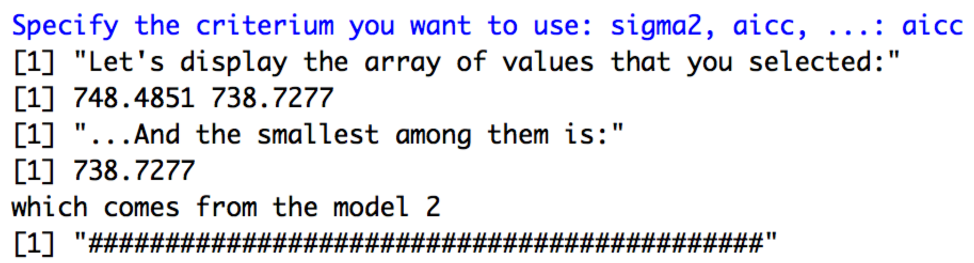
Model comparison

After creating the model matrix, we compare them using some criteria to find out the best model to be used for the further exploratory analysis. Currently, this function only supports to compare models based on the statistical values inside the ARIMA() function in R, for instance, sigma^2, aicc, bic, …, and it only supports 1 criterion.



```
Specify the criterium you want to use: sigma2, aicc, ...: aicc
```

Going forward, the function will automatically display all the specified values from all models and display them in a readable format.

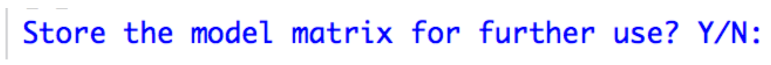


```
Specify the criterium you want to use: sigma2, aicc, ...: aicc
[1] "Let's display the array of values that you selected:"
[1] 748.4851 738.7277
[1] "...And the smallest among them is:"
[1] 738.7277
which comes from the model 2
[1] "############################################"
```

At this point, I haven't figured out a way to let user input the comparing method, and the default method in this function is min(), so it will only display the smallest value among the models. If I have time in the future, I will try to make this part open to users and then they can use any type of comparing method they want.

Save the model matrix

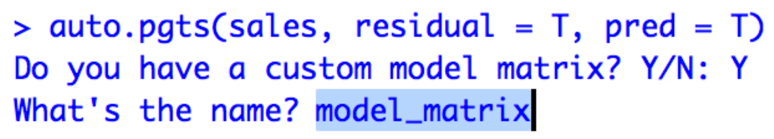
This is more of a breakpoint in the function, it gives the user a-cup-of-coffee time to think about the model matrix they just input. Quite often what will happen is, the user is not satisfied with the model matrix, and they want to modify it, either adding more models in it or changing some numbers. Clearly nobody wants to do it all over again, until he/she input "N". :-)



```
Store the model matrix for further use? Y/N:
```

The model matrix will be stored as a global variable, and feel free to use it next time to save some hassle, because the very first question being asked in the function is if or not the user has a custom matrix.

Okay, so far so good, but what about the residual and predictive analysis? I am not lying, though. Let's rerun this function one more time, and this time, let's use a custom model matrix and specify the two parameters to "T".



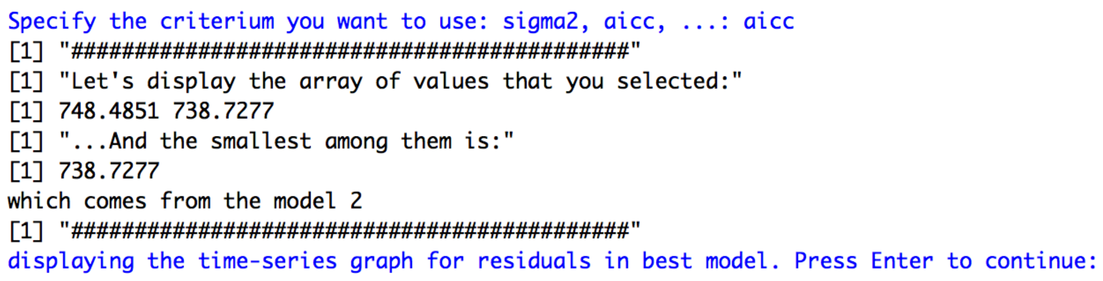
```
> auto.pgts(sales, residual = T, pred = T)
Do you have a custom model matrix? Y/N: Y
What's the name? model_matrix
```

The function will ask users what your custom matrix's name is, and just put in the name that displayed in your "Environment". The default stored name is "model_matrix", unless you have changed it somehow.

Residual analysis

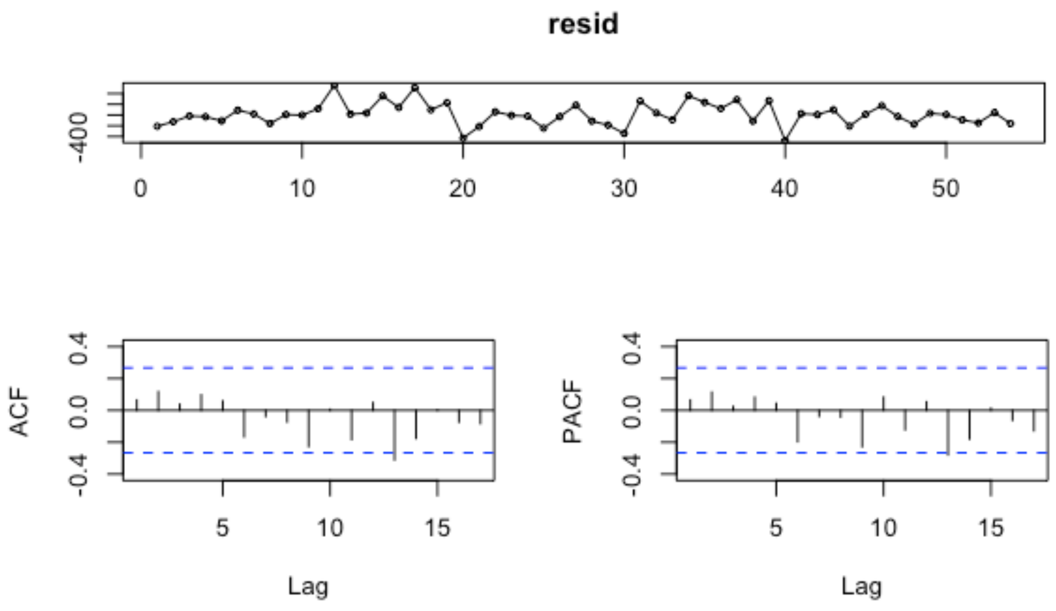
After comparing the model part, users will have a chance to look at the residuals. The overall process is very similar to what we always do in class:

- Look at the time-series data for residuals
- Augmented Dickey-Fuller Test
- KPSS Test for Level Stationarity
- Box-cox test
- Normality test using qqplot()



```
Specify the criterium you want to use: sigma2, aicc, ...: aicc
[1] "#############################################"
[1] "Let's display the array of values that you selected:"
[1] 748.4851 738.7277
[1] "...And the smallest among them is:"
[1] 738.7277
which comes from the model 2
[1] "#############################################"
displaying the time-series graph for residuals in best model. Press Enter to continue:
```

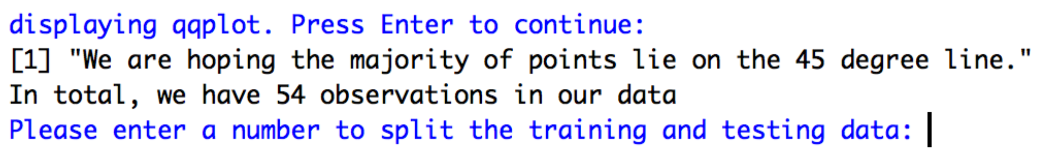
To make sure users are not overwhelmed by the huge amount of information displayed both in the console and the plotting area, I have put in a break point after each major bullet point. The users then need to hit the "Enter" to continue browsing. For example, only after hitting the "Enter" as what is described in the above picture, the following graph will be showed.



With the provided information, users should be empowered to decide whether this analysis is good enough in providing such information for residuals.

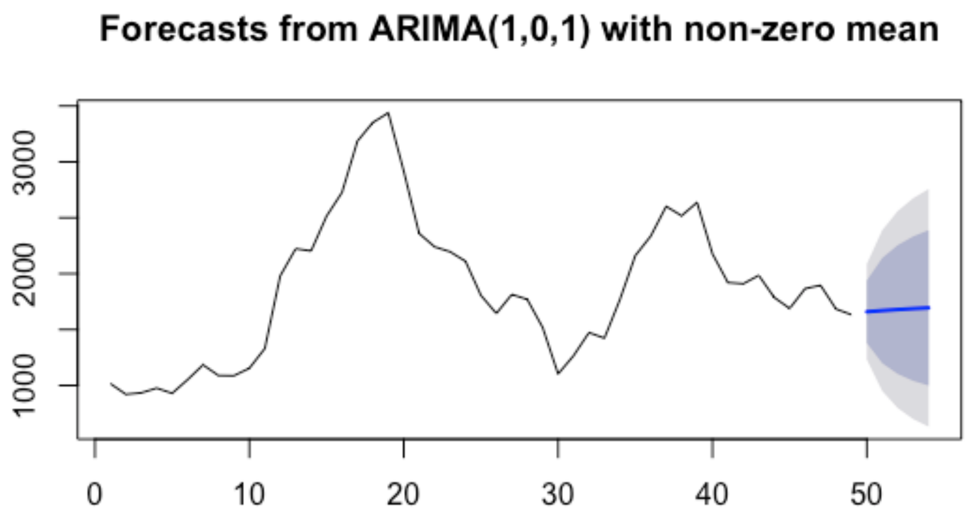
Predictive analysis

After the residual analysis, the function will again stop and wait for the input from user like what is showed in the following picture:

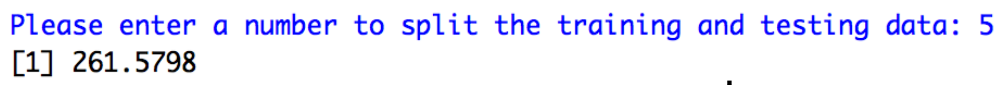


```
displaying qqplot. Press Enter to continue:
[1] "We are hoping the majority of points lie on the 45 degree line."
In total, we have 54 observations in our data
Please enter a number to split the training and testing data: |
```

After the last section in the residual analysis, which is the qqplotting, the user will be notified with the total number of observations in the data, and how the user would like to split up the training and testing data for the predictive analysis. ***It can be somewhat confusing here, but the number you input will be used as the testing data breakpoint, for example, if you input 5, since there are 54 observations, it means the last 5 observations will be used as the testing data, while the first 49 observations will be used as the training data.***



The model being used here, is again, the best model we have found out throughout the process, and there's no need to specify the model again since the function has deliver this piece of knowledge from above. Apart from the predictive plot, the function will also return the RMSE from the testing dataset.



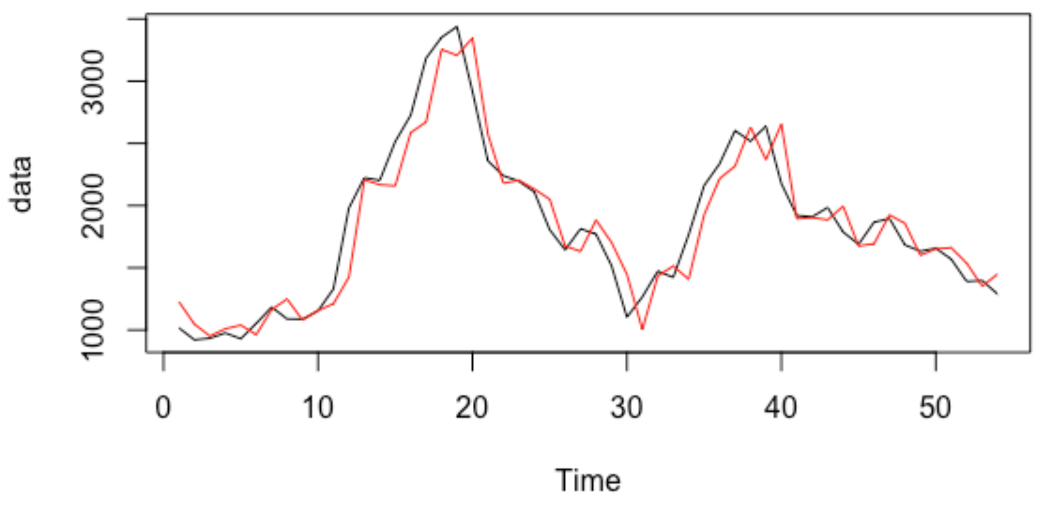
```
Please enter a number to split the training and testing data: 5
[1] 261.5798
```

In the future, I am hoping to extend this section a bit to provide more information to the users.

While it seems like everything is done, this function actually gives the user another perspective to look at the selected model, by applying this model to the whole dataset and plot out the original time-series data trend and the crafted one with the following option:

```
Want to see how well the fitted data is? Y/N: Y
```

If users input “Y”, the following graph will them be showed at last, with the black like being original data and the red line being the fitted data from the best model.



Reaching this point, the function ends with all the above information given to the users.

Conclusion

This is not a perfect function, and it never will be. It is born with the inspiration to help sort out the ARIMA analysis we have gone through in class, and to help non-technical students especially MBA students, to have a tool at hands without having to care too much about the coding side in R.

Reading through the code available in the DSO-522 class, I’ve tried to find out the “secret” behind the ARIMA analysis, and I firmly believe that the manual method taught in class has undisputable advantages compared against the auto ARIMA functions. By looking at the graph and statistical information, users have a chance to regain the power of statistics in their own hands, rather than giving up all technical sides and leaving all done by the computer. This is the right way to study statistics, and credits go to Professor Gabrys.

Appendix

The following code is available for download at https://github.com/flyws/auto.pgts/blob/master/auto.pgts.R

```
auto.pgts <- function(data, residual = F, pred = F){
  if("forecast" %in% rownames(installed.packages()) == FALSE)
    {install.packages("forecast")}
  if("tseries" %in% rownames(installed.packages()) == FALSE)
    {install.packages("tseries")}
  # load packages
  library(forecast)
  library(tseries)

  #choose between exploratory analysis or go directly to customization
  answer0 = readline("Do you have a custom model matrix? Y/N: ")
  if(answer0 == "N"){
    #display the data
    tsdisplay(data)
    # do the stationarity test
    cat(paste(" Augmented Dickey-Fuller Test(p-value <
0.05):",adf.test(data)$p.value),
        "\n",
        paste("KPSS Test for Level Stationarity(p-value >
0.05):",kpss.test(data)$p.value))
    answer1 = readline("Try differencing the data? Y/N: ")
    answer2 = "N"

    # find the right differencing value

    if(answer1 == "Y"){
      while(answer2 != "Y"){
```

```
      diffdata = readline("Please enter your diff data here: ")
      diffdata = eval(parse(text = diffdata))
      tsdisplay(diffdata)
      # display the stationarity test for evaluation
      cat(paste("Augmented Dickey-Fuller Test(p-value <
0.05):",adf.test(diffdata)$p.value),
          "\n",
          paste("KPSS Test for Level Stationarity(p-value >
0.05):",kpss.test(diffdata)$p.value))

      answer2 = readline("Do you like it now? Y/N: ")
      data = diffdata
    }
  }else{
    NULL
  }

  num_of_models = as.integer(readline("How many models you want to try: "))
  model_matrix = matrix(nrow = num_of_models, ncol = 6)
  for(i in 1:num_of_models){
    print(paste("input for model", i))
    cat("Please input two vectors that will be used as orders for model",i)
    num = scan(what = list("",""))
    model_matrix[i,1:3] = eval(parse(text = num[[1]]))
    model_matrix[i,4:6] = eval(parse(text = num[[2]]))
  }

  model = NULL
  criteria = readline("Specify the criterium you want to use: sigma2, aicc, ...: ")
  for(i in 1:num_of_models){
    model = c(model, Arima(data, order = model_matrix[i,1:3],
                  seasonal = model_matrix[i,4:6])[criteria][[1]])
  }

  print("Let's display the array of values that you selected:")
  print(model)
  print("...And the smallest among them is:")
  print(min(model))
  cat("which comes from the model", match(min(model),model),"\n")
  print("############################################")
  answer3 = readline("Store the model matrix for further use? Y/N: ")
  if(answer3 == "Y"){
    model_matrix <<- model_matrix
  }else{
    NULL
  }
 }else{
  #get the custom matrx name
  model_matrix = eval(parse(text = readline("What's the name? ")))
  model = NULL

  #calculate the criteria value
  criteria = readline("Specify the criterium you want to use: sigma2, aicc, ...: ")
  num_of_models = nrow(model_matrix)
  for(i in 1:num_of_models){
    model = c(model, Arima(data, order = model_matrix[i,1:3],
                seasonal = model_matrix[i,4:6])[criteria][[1]])
  }

  print("#############################################")
  print("Let's display the array of values that you selected:")
  print(model)
  print("...And the smallest among them is:")
  print(min(model))
  cat("which comes from the model", match(min(model),model), "\n")
  print("#############################################")
 }

 # Save the best model in the global environment
 best_model <<- Arima(data, order = model_matrix[match(min(model),model),1:3],
           seasonal = model_matrix[match(min(model),model),4:6])
 best_model_index <<- model_matrix[match(min(model),model),]

 # Perform the residual analysis
 if(residual == T){
  readline("displaying the time-series graph for residuals in best model. Press Enter
to continue: ")
  resid = residuals(best_model)
  tsdisplay(resid)

  # do the stationarity test
  cat(paste(" Augmented Dickey-Fuller Test(p-value <
0.05):",adf.test(resid)$p.value),
      "\n",
      paste("KPSS Test for Level Stationarity(p-value >
0.05):",kpss.test(resid)$p.value))

  # Box Cox test
  readline("displaying Box test. Press Enter to continue: ")
  print("Keep in mind that the Ho: lag 1 corr = ... = lag k corr = 0")
  cat("\n")
  print("so we are hoping to p-value to be high to state independece")
  print(Box.test(resid))

  # Normality test
  readline("displaying qqplot. Press Enter to continue: ")
  qqnorm(resid)
  qqline(resid)
  print("We are hoping the majority of points lie on the 45 degree line.")
 }

 if(pred == T){
  cat("In total, we have", length(data), "observations in our data","\n")
  h = readline("Please enter a number to split the training and testing data: ")
  h = as.integer(h)
  rmsets(data, h)

  answer4 = readline("Want to see how well the fitted data is? Y/N: ")
  if(answer4 == "Y"){
    plot(data)
    FC = forecast(best_model, h = h)
    lines(fitted(FC), col = 2)
  }
 }
}

rmsets = function(tsdata, h,...)
{
 train.end=time(tsdata)[length(tsdata)-h]
 test.start=time(tsdata)[length(tsdata)-h+1]
 trainingset=window(tsdata,end=train.end)
 testingset = window(tsdata, start=test.start)
 training_model = Arima(trainingset, order = best_model_index[1:3],
                                 seasonal = best_model_index[4:6])
 modelforecast=forecast(training_model,h=h)
 print(accuracy(modelforecast,testingset)[2,"RMSE"])
  plot(modelforecast)
}
```