The impact of macroeconomic variables on income distribution

Humboldt-Universität zu Berlin
School of Business and Economics
Ladislaus von Bortkiewicz Chair of Statistics
Seminar in Statistical Programming Languanges

by

Lyubomir Danov (561403)
Ana Lluberes (577397)
Björn Viergutz (570140)

Berlin, August 14, 2016

Contents

List of Figures

List of Tables

1 Introduction

The relationship between the income distribution and key macroeconomic variables has been subject to numerous empirical studies. In this paper, we run a time series regression on U.S. data covering a time span from 1967 to 2014. The Gini coefficient serves as an indicator of the income distribution. It is regressed on three key macroeconomic variables: inflation, gross domestic product and unemployment. The aim of this paper is to discuss the validity of this approach. In particular, stationarity is an issue in regression analysis of time series data. That is, the OLS estimator is not consistent when the time series is non-stationary.

In order to account for the non-stationarity that we observe in our variables, e.g. GDP obviously follows a trend, we set up a regression model in terms of first differences. We provide plots that visualize our data in both levels and first differences. Additionally, tests on stationarity can be conducted. Former studies make use of various test such as the Dickey-Fuller or the augmented Dickey-Fuller test. The null hypothesis of these tests is that there is a unit-root. This is often rejected very easily. However, it does not mean that the data is stationary. That is why we also apply the more sensible KPSS test.

A self-implemented version of the Dickey-Fuller test is at the core of our paper. We assess its performance to the ADF test with lag length of 1 for reasons of comparability. In the process with reviewing our approach, we come up with an updated version of our self-conducted Dickey-Fuller test that comes significantly closer to the ADF estimates than a previously conducted original version.

The paper is organized as follows. Section 2 gives a short review on other studies that used the same or a similar aproach. Furthermore, that section focuses on the issue of stationarity in time series regression analysis. Three tests for stationarity are presented; the differences between them as well as the respective test ideas are described. Section 3 describes the dataset. Section 4 covers in detail the specific requirements, goals and limitations of the listed stages of our QuantLet. Section 5 evaluates the performance of the three different tests on stationarity and discusses the empirical results of our study. That section also relates the findings of this paper to the spurious regression problem. Section 6 concludes.

2 Theory and Design

2.1 Model

Former studies, such as Alan S. Blinder (1978), set up a simple regression model using income quintiles as dependent variable and as a measure of income inequality. Using a similar model, Jäntti and Jenkins (2010) examine the effect of growth, inflation, real interest rate and non-employment rate on the Gini coefficient. In line with both approaches, in this paper we suggest a model in which the Gini-coefficient ($GINI_t$) is regressed on three key macroeconomic variables. INF_t denotes the annual inflation rate, GDP_t the real gross domestic product and $UNEMPL_t$ the unemployment rate.

$$GINI_t = \beta_0 + \beta_1 * INF_t + \beta_2 * GDP_t + \beta_3 * UNEMPL_t$$

2.2 Stationary vs. non-stationary variables in time series regression

A *stationary* time series is defined as one whose statistical properties are constant over time, meaning it does not follow any trend, i.e. the variable is described by the exact same distribution at each point in time. Given the strictness of this definition, the term *weak-stationarity* (also referred to as *second order stationarity*) is usually employed, where only the mean and the variance do not depend on t, and the covariance between two values in time t and t - j depends only on j (i.e. the difference between t and t - j and not the location of the points on the time axis). More formally,

$$E[X_t] = \mu \ \forall t$$
$$Var[X_t] = \sigma^2 \ \forall t$$
$$Cov[X_t; X_{t-j}] = \gamma(j) \ \forall t$$

It is only when a time series complies with this assumption that the OLS estimates are consistent, as shown by Granger and Newbold (1974) in their study on *spurious regressions*. When a time series fails to satisfy the requirements stated above, it is said to be *non-stationary*.

Given that macroeconomic time series data usually follows a trend, or some other sort of cyclical or seasonal behavior, when working with variables such as gross domestic product,

unemployment rates and inflation rates it is expected that the stationarity issue will arise, considering the non-stationary nature of such data.

Following this intuition, prior to running any regression model, stationarity on both the regressands and regressors needs to be evaluated. First, we consider the graphical time series behavior in levels and first differences, as suggested by Brockwell and Davis (2013). Additionally, following Mocan (1999) and Jäntti and Jenkins (2010), we conduct stationarity tests on both level data and first differences. Mocan (1999) emphasizes the inappropriateness of using level data which is why he suggests using a regression model based on first differences.

With the Dickey-Fuller Test (DF) and the Augmented Dickey-Fuller Test (ADF) we can test for a unit root in a time series. The null hypothesis of both tests is that there is a unit root in the following autoregressive model:

$$\Delta y_t = (\rho - 1)y_{t-1} + u_t = \delta y_{t-1} + u_t$$

There are three sensible versions of the test (no constant, constant, constant and time trend). We also implement the Kwiatkowski-Phillips-Schmidt-Shin (KPSS), as complementary test. Unlike the DF and the ADF test, the null hypothesis of the KPSS test is that the time series is (trend-) stationary, and the alternative hypothesis is that the time series has a unit root.

3 Data

3.1 Dataset

We take time series data from the Federal Reserve Bank of St. Louis which is available on Thomson Reuters Datastream. It contains 48 yearly observations of the following economic indicators:

Variable	Explanation
GDP_t	real US gross domestic product
INF_t	inflation rate
$UNEMPL_t$	unemployment rate
$GINI_t$	Gini coefficient

Table 1: Variables in US time series data

The data covers a time span from 1967 to 2014 and is thus a very recent dataset. Throughout this period, the U.S. economy suffered from several shocks and crises: the Vietnam War, the increase of oil prices in 1990 and 2003 due to the Gulf war and the war in Iraq, respectively, the dotcom bubble burst at the beginning of the new millennium and the financial crisis of 2008. To sum up, the time period that we look at includes disturbances that may have had an impact on income inequality.

3.2 Time series plots in levels

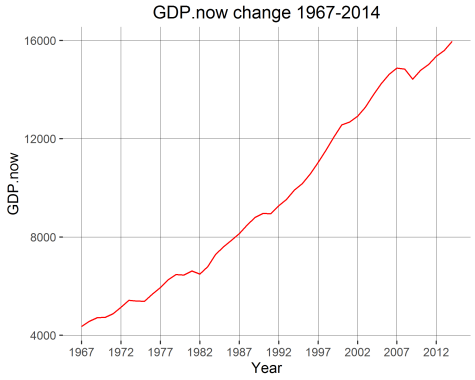

Figure 1: GDP levels

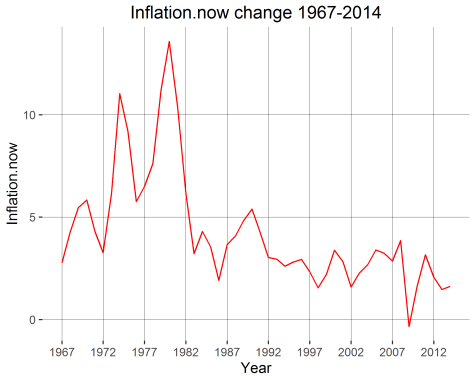

Figure 2: Inflation levels

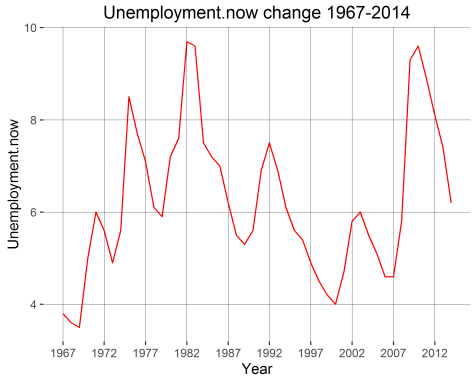

Figure 3: Unemployment levels

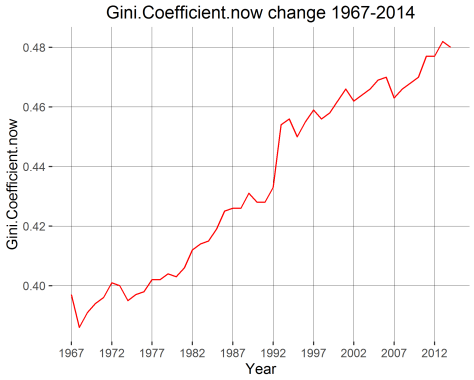

Figure 4: Gini Coefficient levels

3.3 Time series plots in first differences

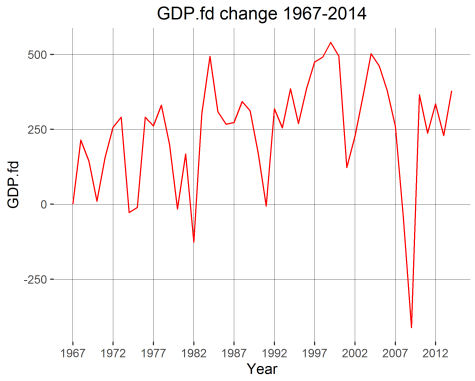

Figure 5: GDP first differences

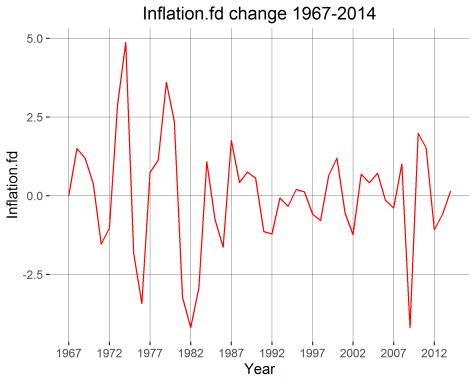

Figure 6: Inflation first differences

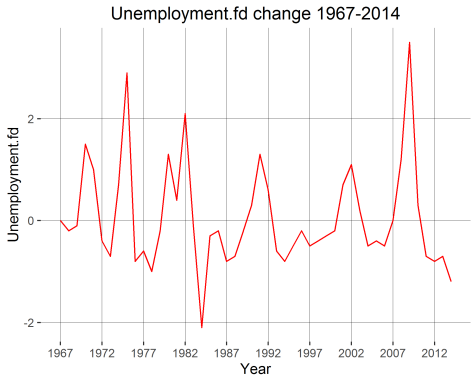

Figure 7: Unemployment first differences

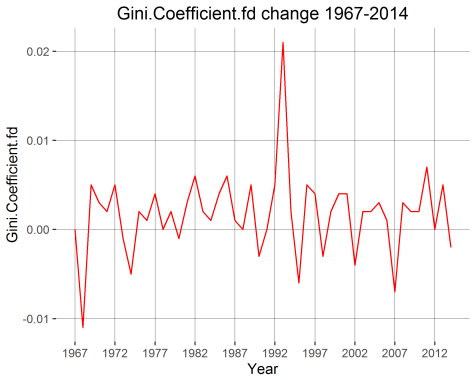

Figure 8: Gini coefficient first differences

3.4 Graphical analysis of the time series behaviour

	Levels	First differences
GDP_t	There is a constant increase of the GDP, revealing an almost linear, **positive trend**. From the graph, we can infer **non-stationarity**. With respect to economic theory, a constant GDP growth is absolutely realistic.	The values fluctuate. **No trend** seems to be present, the variable appears to be **stationary**.
INF_t	No specific conclusion can be made regarding stationarity from observing the time series behaviour. It appears to follow a slight downward trend after 1980, but it's not obvious.	The values fluctuate heavily around a mean value, zero. **No** sign of a **trend** is visible, the time series could be regarded as **stationary**.
$UNEMPL_t$	Unemployment rates dont seem to follow a specific trend behavior. Nevertheless, no obvious stationary mean value is observable. **No trend** and **non-stationarity** could be inferred.	The values appear less disperse than in levels. Considering the observable stability within the fluctuations of the data, the variable seems to follow a **stationary** process, **without** any visible **trend**
$GINI_t$	The variable shows a strong almost linear **positive trend**. From the graph, it seems clear that the variable is **non-stationary**.	The positive tendency is no longer observable in the graph of first differences. The variable appears to be **stationary** with **no trend**.

Table 2: Time series behavior

4 Implementation

4.1 Code structure

This section of the paper will cover in detail the specific requirements, goals and limitations of the listed stages of our QuantLet. Initially, we presented our code structure according to Figure 9. In the process of updating the code and enhancing it, a new structure was adopted,

shown in Figure 10. The program is initiated with the SPL_RAwMTSD.R code. Its sole purpose is to convene the user by automatically sourcing each of the stages of our code.

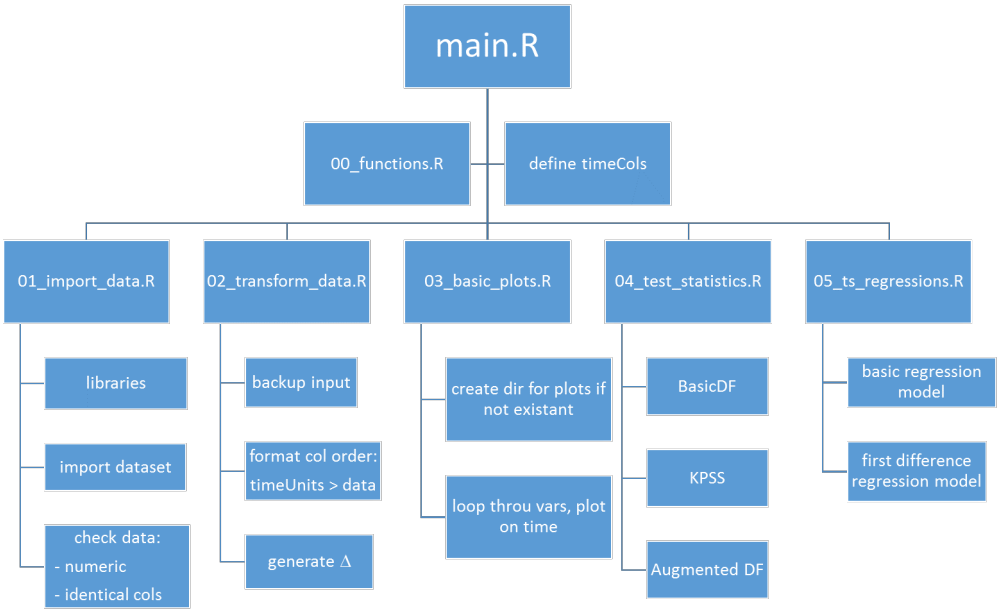

Figure 9: Original organization of the code

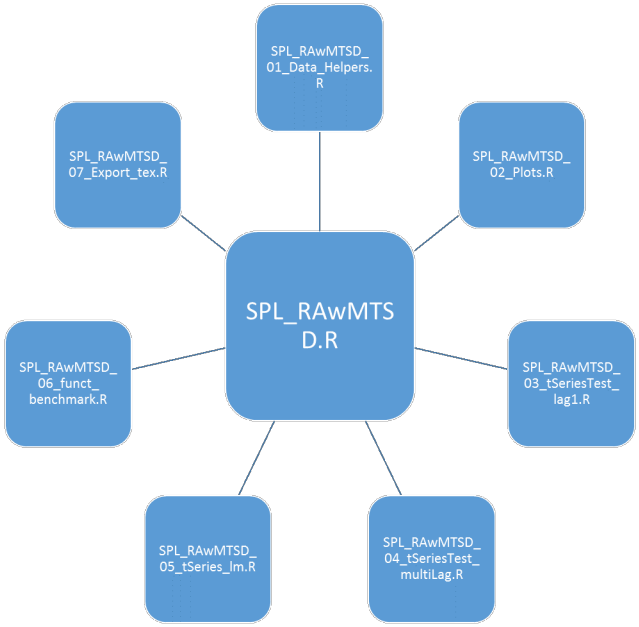

Figure 10: Updated organization of the code

4.2 Script: SPL_RAwMTSD_01_Data_Helpers.R

Requirements

- Function source codes available in working directory (WD) subfolder named /sources
- Data present in WD, in our projects case SPL_RAwMTSD_data_v2.csv
- Users input on timeCols a vector with the names of columns containing index and time unit data (year, month, etc.)

Goals

The goals of this script are to load project specific functions, load the required dataset, generate first differences on all non-index columns and rearrange columns in object input of class data.frame for readability.

Limitations

The script is limited in expecting a specific format of the input dataset (SPL_RAwMTSD_data_v2.csv). This effect is offset by some of the helper functions loaded at its start:

- PrepPackages function takes as input a character vector of package names, automatically installs them, if not found by installed.packages function (limited by library paths), loads the packages and suppresses most startup messages.
- ColsSame function compares the first column from a data.frame to all subsequent columns. It takes a vector of column indexes and a data object as arguments. It is limited by having the user define a separate column to compare against all others.
- ConvertNumeric expands on Rs build-in functions to automatically try to convert a given column to numeric. This is the case when read.table (and its implementations read.csv and read.csv2) does not use arguments, adjusted to the specific dataset called. The function allows for thousand and decimal separators to be defined after the data object has been loaded into the global environment. The user can specify on which columns the attempted conversion will take place. It also keeps the original columns if the output is not as intended.

4.3 Script: BasicDF.R and BasicDFv2.R

BasicDF.R contains the initial version of our implementation of the Dickey-Fuller (DF) Test: basic.DF.old() function. It takes as arguments a data object and the names of vectors, for which three levels of the DF test will be generated:

- Regressing first differences of variable on t-1 value of the variable
- As previous, but including an intercept
- As previous, but also including a time index variable

The resulting object is a data frame with each row representing the results of the test lag variables beta and t-value and the regressions p-value, for each of the three levels of the test.

In the course of presenting our implementation, questions were raised about the differences in results between this DF implementation and the Augmented DF from the tseries package with a lag of 1. The differences in p-values (Table 4) were only for certain variables. After investigating, the following was found: the adf.test() function from the tseries package uses a build in table for approximating the p-value of the lag variable, while our function was reporting the p-values of the regression. The adf.test() function also reports the t-value of the lag variable as the test statistic, while we, due to the intuitive nature of interpretation of the test as a result of a regression, had decided to report the beta coefficient of the lag variable as well.

BasicDFv2.R contains our updated implementation of the DF test: basic.DF(). As well as addressing the differences in results: it now reports the p-value of the coefficient, as well as the one of the regression. As it can be seen in Table 5, there are still differences when the adf.test() function reports p-values significantly above a 0.10 threshold, while we report the p-values generated by the lm() function.

Our reiterated implementation now also allows the user to decide the levels, on which the DF unit root test is to be performed (Base, Base with drift (called Drift), Complete, All or any combination of them). In the course of development it was found, that using rbind() between a data frame and a vector could lead to introduction of NA values, while the function seems to not respect the stringsAsFactors = FALSE argument.

4.4 Script: SPL_RAwMTSD_02_Plots.R

Requirements

- A data.frame object input as generated by SPL_RAwMTSD_01_Data_Helpers.R
- Users input on x scale (in our case input$Year) and indices of data columns
- Loaded ggplot2, foreach and doParallel packages

Goals

The goals of this script is to quickly and automatically generate identically styled graphs of data variables in the dataset, focusing the time on analysis instead of code development and testing. The generated plots are of the same size and in the same automatically created subfolder /plots

Limitations

The script is theoretically limited by the limitations of the foreach() and ggplot() functions. Creating multiple graphs can be slow on certain systems. To counter such performance the %dopar% operator of the foreach() function was used, creating a separate R session for each core available (automatically taking the number of cores with the Sys.getenv() function).

4.5 Script: SPL_RAwMTSD_03_tSeriesTest_lag1.R

Requirements

- A data.frame object input as generated by SPL_RAwMTSD_01_Data_Helpers.R
- Loaded tseries package and sourced custom BasicDFv2.R script

Goals

The goal of this script is to quickly and automatically generate identically styled data frames, containing the results of DF, ADF (with alternative hypothesis both explosive and stationary) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test (with null hypothesis both level and trend stationarity) with a lag value of 1. For this purpose, it generates 5 data frames, one for each hypothesis.

Limitations

The script is limited by the use of the base::for() function in two ways: first, as will be covered by the benchmarking section and script, it is the slowest of the three (other two being apply-like sapply and lapply and foreach). Second, it assumes that within the data frame input, the first and second column contain time indices and as such generates the three tests for each column after the 3rd (but iterates over each subsequent column). Limitations of the used adf.test() and kpss.test() functions is the return of a list object, which is difficult to analyze with an increasing number of variables, so the results are converted into data frames. Column names have to be well defined, as they serve as identifiers in the data frames, containing the results. Another limitation is the lag value of 1 in the KPSS test implementation: KPSS automatically assumes a lag value from either of the following two formulas:

$$Lshort = trunc(3 * sqrt(n)/13)$$
$$Llong = trunc(10 * sqrt(n)/14)$$

4.6 Script: SPL_RAwMTSD_04_tSeriesTest_multiLag.R

Requirements

- A data.frame object input as generated by SPL_RAwMTSD_01_Data_Helpers.R
- Loaded tseries package

Goals

The goals of this script are to quickly and automatically generate identically styled data frames, containing the results of ADF (with alternative hypothesis both explosive and stationary) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test (with null hypothesis both level and trend stationarity). As will be covered in the limitations section of this script, the kpss.test() function was customized to accept more than two automatically generated lag values. The script results in 3 data frames, one for each test, and each including both available hypothesis options, and lag values from 1 to 21. A maximum lag of 21 was chosen as the adf.test() function returned errors for greater values.

Limitations

As with the previous script, this one is as well limited by the use of the base::for() function in the same two ways: speed and predetermined input dimensions. To limit the speed impact,

the foreach::foreach() function was used in combination with the %dopar% operator and registerDoParallel(). The limitations of the adf.test() and kpss.test() functions are also the same with some additions:

- adf.test() could only generate the test for lag values of up to 21.
- kpss.test() was used with the resulting lag value of the Long lag formula (where lag = 4)

Due to the limitations of tseries kpss.test, a customized version was also introduced, allowing for user-determined lag levels. This might have the effect of faultily reported p-values and test statistic, as the function uses an approximation on predetermined tables for p-value and standard deviation.

4.7 Script: SPL_RAwMTSD_05_tSeries_lm.R

Requirements

- A data.frame object input as generated by SPL_RAwMTSD_01_Data_Helpers.R
- Loaded tseries package

Goals

The goals of this script are to estimate a linear model of the Gini coefficient based on GDP, Inflation and Unemployment. The lm() function is used on an unaltered dataset for the variables and their first differences. The residuals are plotted and tested with KPSS. A toy logit model is also implemented, but not presented.

Limitations

The script is limited by the predetermined formula and given dataset. Users are able to easily make adjustments to the formula in the script file, but processing the variables might not be as easy a task.

4.8 Script: SPL_RAwMTSD_06_funct_benchmark.R

Requirements

- A data.frame object input as generated by SPL_RAwMTSD_01_Data_Helpers.R
- Loaded tseries, foreach, doParallel, microbenchmark packages

Goals

The goals of this script are to present the differences in performance between base::for(), foreach::foreach with both %do% and %dopar% operators and the apply-like functions lapply and sapply. To do this, the script SPL_RAwMTSD_04_tSeriesTest_multiLag.R, which contains the most data modification and variable generation operations was used by adjusting the script to also use foreach and s-/lapply.

Limitations

The script is mostly limited by the system configuration and hardware. Benchmarking tests were done on a system with an AMD E-350 Processor (2 x 1.6GHz), x64 version of R 3.3.1 and memory limit set to 5736MB. Foreach is tested on one and two cores separately. Each benchmarking script was run 20 times a summary statistics are reported in seconds.

expr	min	lq	mean	median	uq	max	neval
foreach_2CPUs	6.969	7.055	7.547	7.269	8.024	9.287	20
apply	7.108	7.433	7.837	7.751	8.119	8.971	20
foreach_1CPU	10.196	10.513	11.198	11.156	11.771	12.625	20
for	9.854	11.012	11.779	11.788	12.405	14.451	20

Table 3: Benchmark results

4.9 Script: SPL_RAwMTSD_07_Export_tex.R

Requirements

- Input objects as generated by each of the previous scripts
- Loaded stargazer package

Goals

The goals of this script is to generate *.tex versions of specific tables, as well as all generated objects en masse.

Limitations

The script is limited by limitations of the stargazer() function, which, for example, by default assumes a summary of the object. The stargazer() has some limitations in formatting the output, but generates ready to implement tex code and a quickly overviewable *.txt version.

5 Empirical Study and Testing

5.1 Testing for Stationarity: Dickey-Fuller test and augmented Dickey-Fuller test

We apply the original Dickey-Fuller (DF) test and the Augmented Dickey-Fuller (ADF) test on the time series data both in levels and first differences.

As described in Section 4, we conducted the DF test on our own (original implementation). However, the results were not consistent with the results from the ADF test, which is why we reviewed our approach and created an updated implementation. Table 4 shows the results from the original implementation, whereas Table 5 shows the updated results. We use a lag length of 1 for the ADF test in order to make the results comparable.

In the original DF implementation, the p-values differed substantially from those we got from the ADF (Table 4). This raised some doubts about our approach, since they should be equal to the p-values from the ADF test with a lag length of 1.

We conclude that the original implementation is very poor, given that it rejects the null hypothesis for GDP and GINI level data. That is, the test suggests that GDP and GINI level data is stationary. However, the graphical analysis (Table 2) clearly rejects this idea, and so does the ADF.

	DF		ADF	
	Level	F.Diff	Level	F.Diff
GDP_t	0.044	0.003	0.590	< 0.010
INF_t	0.032	< 0.010	< 0.010	< 0.010
$UNEMPL_t$	0.014	< 0.010	0.034	< 0.010
$GINI_t$	0.009	< 0.010	0.590	< 0.010

Table 4: p-values from the original DF implementation vs. ADF

Following Section 4, the updated DF implementation yields results that are considerably

	DF		ADF	
	Level	F.Diff	Level	F.Diff
GDP_t	0.407	< 0.010	0.590	< 0.010
INF_t	< 0.010	< 0.010	< 0.010	< 0.010
$UNEMPL_t$	0.034	< 0.010	0.034	< 0.010
$GINI_t$	0.401	< 0.010	0.590	< 0.010

Table 5: p-values from the updated DF implementation vs. ADF

more consistent with the ADF. Table 5 shows that p-values of the updated implementation differ only slightly with the p-values from the ADF, specifically in the case of GDP and GINI level data. This small differences still occur because we use the "Base" version of the DF test (i.e. testing for stationarity only). In contrast, the p-values from the ADF correspond to the *explosive* version of the test, following the time series behaviour described in Table 2.

5.2 Testing for Stationarity: KPSS test

Another tool for testing whether a time series follows a stationary process or not are provided by two sensible Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. The first specification has trend stationarity (values fluctuate around a trend) as null hypothesis, the second one level stationarity (values fluctuate around a constant mean). Based on our analysis in Table 2 we choose the suitable test specification.

Variable	Levels	First Diff.
GDP_t	< 0.010	0.086
INF_t	< 0.010	0.100
$UNEMPL_t$	0.100	0.100
$GINI_t$	< 0.010	0.100

Table 6: p-values from KPSS test

Table 6 includes the p-values of the KPSS test. We can reject the null hypothesis of stationarity for all variables in levels except for $UNEMPL_t$. As to the data in fist differences, we cannot reject the null of stationarity for all variables.

The results from the KPSS are in line with the results from the DF and ADF with respect to the first differences. There is a discrepancy in the case of INF_t in level data. Both the

DF and ADF reject the null hypothesis for this variable, meaning that INF_t in levels is stationarity. In contrast, the KPSS test also rejects the null hypothesis for this variable, that is it has a unit root.

Taking all the test results into account, we set up the regression model in terms of first differences.

5.3 Empirical Analysis and Discussion of the Results

Table 7 shows the regression output for both level data (1) and first differences (2). We find that none of the chosen macroeconomic variables have a significant effect on the income distribution. For none of the respective estimated coefficients we can reject the hypothesis that they are significantly different from zero. None of p-values is smaller than 0.1. That is, we reject the hypotheis at any conventional significance level. This result is partially in line with previous findings. For example, Jäntti and Jenkins (2010) suggest that neither inflation nor unemployment have significant effects on income inequality. Their regression model is also set up in first differences in order to deal with stationarity. Thurow (1970) finds that general growth results in higher incomes, but no impact on their relative incomes in income quintiles. Our paper shows that these findings from previous studies do not alter when extending the time span.

Interestingly, running the models with level data shows a clear relationship between macroeconomic factors and the U.S. income distribution. For instance, inequality as measured by the Gini-coefficient seems to increase when GDP increases. Vice versa, higher inflation seems to result in a reduction in inequality. However, as emphasized in Section 2.2, this results can be misleading due to non-consistency of the OLS estimators.

This eveolves to a more general problem that econometricians face, namely the spurious regression. One can run a regression on two variables that have obviously nothing to do with each other, because they are almost independent from each other in theory. Or - and this is the most extreme case - two random walks, that have been generated separately from each other. Nevertheless, running a regression will reveal a linear correlation between the two variables although it does not exist. That is why one has to interpret always carefully, especially when talking about causal effects.

	Gini.Coefficient.now	Gini.Coefficient.fd
	(1)	(2)
GDP.now	0.00001***	
	(0.00000)	
Inflation.now	-0.001***	
	(0.0004)	
Unemployment.now	-0.0002	
	(0.001)	
GDP.fd		-0.00000
		(0.00001)
Inflation.fd		-0.001
		(0.0004)
Unemployment.fd		-0.0005
		(0.001)
Constant	0.371***	0.002
	(0.005)	(0.002)
Observations	48	48
R^2	0.961	0.048
Adjusted R^2	0.959	-0.017
Residual Std. Error (df = 44)	0.006	0.005
F Statistic (df = 3; 44)	363.172***	0.740
Note:		*p<0.1; **p<0.05; ***p<0.01

Table 7: Regression output

6 Conclusion

The goal of this paper is to assess the impact of macroeconomic variables on the income distribution, measured by the Gini coefficient. For this purpose, we run a time series regression on U.S. data covering a time span from 1967 to 2014. We find that none of the regressors, namely inflation, gross domestic product and unemployment has a statisticaly significant impact on the income distribution. This finding is in line with previous studies, that use similar approaches.

Running a regression on time series data necessarily raises the issue of stationarity. If the time series follows a non-stationary process, the OLS estimator is not consistent. We validate this approach by tackling the issue of stationarity from four different angles: plotting and graphical detection, Dickey-Fuller test, Augmented Dickey-Fuller Test and KPSS test.

We implemented the Dickey-Fuller test on our own and compared its performance to the ADF test with lag length of 1 for reasons of comparability. An updated version of our self-conducted Dickey-Fuller test came significantly closer to the ADF estimates than a previously conducted original version. We explain the remaining small differences with the three sensible specifications in which our DF test differed from the ADF test.

However, the additionally conducted KPSS is most sensible to stationarity, and perfectly in line with what we concluded from the graphical analysis of the data plots. The KPSS test clearly suggests ussing first difference instead of level data. That is why we decide to run the regression with first differences.

As we find that the impact of the chosen macroeconomic variables is not significant, one could take into consideration other macroeconomic (interest rates) as well as microeconomic factors (education) in order to better account for income inequality.

References

Alan S. Blinder, H. Y. E. (1978): "Macroeconomic Activity and Income Distribution in the Postwar United States," *The Review of Economics and Statistics*, 60, 604–609.

Brockwell, P. J. and R. A. Davis (2013): *Time series: theory and methods*, Springer Science & Business Media.

Granger, C. W. and P. Newbold (1974): "Spurious regressions in econometrics," *Journal of econometrics*, 2, 111–120.

Jäntti, M. and S. P. Jenkins (2010): "The impact of macroeconomic conditions on income inequality," *The Journal of Economic Inequality*, 8, 221–240.

Mocan, H. N. (1999): "Structural unemployment, cyclical unemployment, and income inequality," *Review of Economics and Statistics*, 81, 122–134.

Thurow, L. C. (1970): "Analyzing the American Income Distribution," *The American Economic Review*, 60, 261–269.

7 Appendix

SPL_RAwMTSD.R

```
setwd()
quants.sources = list.files(getwd(), pattern = "\\.r$", ignore.case = TRUE)
print(quants.sources)

source("SPL_RAwMTSD_01_Data_Helpers.R")
source("SPL_RAwMTSD_02_Plots.R")
source("SPL_RAwMTSD_03_tSeriesTest_lag1.R")
source("SPL_RAwMTSD_04_tSeriesTest_multiLag.R")
source("SPL_RAwMTSD_05_tSeries_lm.R")
source("SPL_RAwMTSD_06_funct_benchmark.R")
source("SPL_RAwMTSD_07_Export_tex.R")
```

SPL_RAwMTSD_01_Data_Helpers.R

```
#### Clear Global Environment ####
rm(list = ls())

#### Load functions ####
cd = getwd()
cd.new = paste0(cd, "/sources")
setwd(cd.new)

function.sources = list.files(getwd(), pattern = "\\.r$", ignore.case = TRUE)
for (i in 1:length(function.sources)) {
source(function.sources[i])
}

setwd(cd)
rm(cd, cd.new, i, function.sources)

#### Load libraries ####

### Function PrepPackages Load needed libraries from a vector If not in
    installed.packages, install them xtable
### for df to latex table, stargazer for lm results
PrepPackages(c("tseries", "foreach", "doParallel", "microbenchmark", "ggplot2",
     "stargazer"))
```

```
#### Import ####

### Import dataset
if (!exists("input")) {
input = read.csv("SPL_RAwMTSD_data_v2.csv", header = TRUE, stringsAsFactors =
    FALSE)
print("input")
str(input)
input$timeIndex = c(1:nrow(input))

} else {
print("input")
str(input)
}

#### Dataset check ####

### Function ColsSame Checks if the first column in a vector of column indexes
    is equal to any of the others Was
### developed to keep track of multiple time units columns (quarters, years)
    Returns a list with vectors of column
### indexes - same and differing

### Example use exa = cbind(c(1:25), c(1:25), c(1:25), c(25:1), c(11:35))
    colnames(exa) = c('one', 'two',
### 'three', 'four', 'five') ColsSame(exa) ColsSame(exa, c(3:5, 1, 2)) rm(exa)

if (!is.null(unlist(ColsSame(input)["colsSame"]))) {
toRem = unlist(ColsSame(input)["colsSame"])
input = input[, -toRem]
}

### Function ConvertNumeric Replaces thousand and decimal separator with R
    defaults Converts vector to numeric
### Suffix to transformed colnames adjustable Developed to be able to transform
     R objects after import

### Example use test = read.delim(paste0(getwd(), '/demo/test_for_num_function.
    csv'), sep = ';', stringsAsFactors
```

```
### = F) str(test) test1 = ConvertNumeric(test, thSep = '_', decSep = ',',
    colSuffix = '') str(test1) rm(test,
### test1)

#### Define time variable and indices column names
timeCols = c("Year", "timeIndex")

#### Generate first differences ####
fDcols = colnames(input)[!colnames(input) %in% timeCols]

for (i in 1:length(fDcols)) {
colnames(input)[colnames(input) == fDcols[i]] = paste0(fDcols[i], ".now")
fDcols[i] = paste0(fDcols[i], ".now")
}

foreach(j = 1:length(fDcols)) %do% {

k = ncol(input) + 1
input[, k] = 0

for (i in 2:nrow(input)) {
input[i, k] = input[i, fDcols[j]] - input[i - 1, fDcols[j]]
}

colnames(input)[k] = gsub(".now", ".fd", fDcols[j])

}

#### Rearrange columns ####
colArrangement = timeCols

for (i in 1:length(fDcols)) {
colArrangement = c(colArrangement, fDcols[i], gsub(".now", ".fd", fDcols[i]))
}

input = input[, colArrangement]
rm(colArrangement, i, j, k)
```

BasicDF.R

```
basic.DF.old = function(data, fDcols = colnames(data)) {

library(foreach)
### Generate 3 variants of DF test
DF_values = data.frame(matrix(vector(), nrow = 1, ncol = 5), stringsAsFactors =
    FALSE)
colnames(DF_values) = c("varName", "test.level", "test.stat", "tval.test.stat",
    "pval.regr")

foreach(i = 1:length(fDcols)) %do% {
y.now = data[, fDcols[i]]
y.diff = rep(0, length(y.now))

for (j in 2:length(y.now)) {
y.diff[j] = y.now[j] - y.now[j - 1]
}



y.prev = y.now[-length(y.now)]
y.diff = y.diff[-1]
timeTrend = c(1:(length(y.prev)))
ur = lm(y.diff ~ y.prev - 1)
ur.dr = lm(y.diff ~ y.prev)
ur.dr.tt = lm(y.diff ~ y.prev + timeTrend)

out.ur = c(fDcols[i], "UR.base", summary(ur)$coefficients[1], summary(ur)$
    coefficients[3], pf(summary(ur)$fstatistic[1],
summary(ur)$fstatistic[2], summary(ur)$fstatistic[3], lower.tail = F))
out.ur2 = c(fDcols[i], "UR.drift", summary(ur.dr)$coefficients[2], summary(ur.
    dr)$coefficients[6], pf(summary(ur.dr)$fstatistic[1],
summary(ur.dr)$fstatistic[2], summary(ur.dr)$fstatistic[3], lower.tail = F))
out.ur3 = c(fDcols[i], "UR.drift.tTrend", summary(ur.dr.tt)$coefficients[2],
    summary(ur.dr.tt)$coefficients[8],
pf(summary(ur.dr.tt)$fstatistic[1], summary(ur.dr.tt)$fstatistic[2], summary(ur
    .dr.tt)$fstatistic[3],
lower.tail = F))
```

```
# temp = list(ur, ur.dd, ur.dd.tt) names(temp) = c(paste0(fDcols[i], '_DF_ur'),
    paste0(fDcols[i], '_DF_urdr'),
# paste0(fDcols[i], '_DF_urdrtt')) assign(paste0('DF_', fDcols[i]), temp)


DF_values = rbind(DF_values, out.ur, out.ur2, out.ur3)

}


DF_values = DF_values[-1, ]
return(DF_values)
}
```

BasicDFv2.R

```
basic.DF = function(data, fDcols = colnames(data), meth = c("All", "Base", "
    Drift", "Complete")) {

DF_values = data.frame(matrix(vector(), nrow = 0, ncol = 6, dimnames = list(
    NULL, c("varName", "test.level",
"test.stat", "tval.test.stat", "pval.coeff", "pval.regr"))), stringsAsFactors =
    FALSE)

chosenSettings = data.frame(matrix(c(0), nrow = 1, ncol = 3, dimnames = list(
    NULL, c("Base", "Drift", "Complete"))),
stringsAsFactors = FALSE)

if (TRUE %in% grepl("All", meth, ignore.case = TRUE)) {
chosenSettings[1, ] = 1
} else {
for (k in 1:ncol(chosenSettings)) {
if (TRUE %in% grepl(colnames(chosenSettings)[k], meth, ignore.case = TRUE)) {
chosenSettings[1, colnames(chosenSettings)[k]] = 1
}
}
}


for (i in 1:length(fDcols)) {
```

```
if (length(fDcols) == 1) {
y.now = as.numeric(data)
} else {
y.now = data[, fDcols[i]]
}

y.diff = diff(y.now)
y.prev = y.now[-length(y.now)]
timeTrend = c(1:(length(y.prev)))

if (chosenSettings[1, 1] == 1) {
ur = lm(y.diff ~ y.prev - 1)
out.ur1 = data.frame(fDcols[i], "UR.base", summary(ur)$coefficients[1], summary
    (ur)$coefficients[3],
summary(ur)$coefficients[4], pf(summary(ur)$fstatistic[1], summary(ur)$
    fstatistic[2], summary(ur)$fstatistic[3],
lower.tail = F), stringsAsFactors = FALSE)
colnames(out.ur1) = c("varName", "test.level", "test.stat", "tval.test.stat", "
    pval.coeff", "pval.regr")
}

if (chosenSettings[1, 2] == 1) {
ur.dr = lm(y.diff ~ y.prev)
out.ur2 = data.frame(fDcols[i], "UR.drift", summary(ur.dr)$coefficients[2],
    summary(ur.dr)$coefficients[6],
summary(ur.dr)$coefficients[8], pf(summary(ur.dr)$fstatistic[1], summary(ur.dr)
    $fstatistic[2], summary(ur.dr)$fstatistic[3],
lower.tail = F), stringsAsFactors = FALSE)
colnames(out.ur2) = c("varName", "test.level", "test.stat", "tval.test.stat", "
    pval.coeff", "pval.regr")
}

if (chosenSettings[1, 3] == 1) {
ur.dr.tt = lm(y.diff ~ y.prev + timeTrend)
out.ur3 = data.frame(fDcols[i], "UR.drift.tTrend", summary(ur.dr.tt)$
    coefficients[2], summary(ur.dr.tt)$coefficients[8],
summary(ur.dr.tt)$coefficients[11], pf(summary(ur.dr.tt)$fstatistic[1], summary
    (ur.dr.tt)$fstatistic[2],
summary(ur.dr.tt)$fstatistic[3], lower.tail = F), stringsAsFactors = FALSE)
colnames(out.ur3) = c("varName", "test.level", "test.stat", "tval.test.stat", "
    pval.coeff", "pval.regr")
```

```
}

DF_values = rbind(DF_values,
if (exists("out.ur1")) {
out.ur1
},
if (exists("out.ur2")) {
out.ur2
},
if (exists("out.ur3")) {
out.ur3
}
)

}

return(DF_values)
}
```

SPL_RAwMTSD_02_Plots.R

```
### First plots

### Create plots in specified directory. If directory does not exist, create
    it.

if (!dir.exists(paste0(getwd(), "/plots"))) {
print("Creating folder '/plots'")
dir.create(paste0(getwd(), "/plots"))
} else {
print("Folder '/plots' exists. Careful to not overwrite existing plots")
}

clst = makeCluster(as.numeric(Sys.getenv("NUMBER_OF_PROCESSORS")))
registerDoParallel(clst)

foreach(i = 3:ncol(input), .packages = "ggplot2") %dopar% {
if (!grepl(".prev", colnames(input)[i])) {
plotPath = file.path(paste0(getwd(), "/plots/"))
```

```
temp = ggplot(input, aes(Year)) + geom_line(aes(y = input[, i]), colour = "red"
    , size = 0.5) + scale_x_continuous(breaks = seq(min(input$Year),
max(input$Year), by = 5)) + labs(title = paste0(colnames(input[i]), "_change_",
     min(input$Year), "-",
max(input$Year)), x = "Year", y = colnames(input[i])) + theme(panel.background
    = element_rect(fill = "white"),
panel.grid.major = element_line(colour = "black", size = 0.15))

ggsave(filename = paste0(colnames(input)[i], ".png"), path = plotPath, plot =
    temp, width = 5, height = 4,
dpi = 400)
rm(temp)
}
}

stopCluster(clst)
```

SPL_RAwMTSD_03_tSeriesTest_lag1.R

```
#### Dickey-Fuller Test ####
DF_base = basic.DF(input, colnames(input)[3:10], meth = "All")
# DF_base_old = basic.DF.old(input, colnames(input)[3:10])

#### Augmented Dickey-Fuller Test ####
ADF_stationary = data.frame(matrix(vector(), nrow = 0, ncol = 6))
ADF_explosive = ADF_stationary

### Generate Augmented Dickey-Fuller Test statistic with H1: stationarity and
    lag 1
for (i in 3:length(input)) {
if (grepl(".prev", colnames(input)[i])) {
(next)()
}
temp = adf.test(input[, i], alternative = "s", k = 1L)
temp = t(unlist(temp))
temp[, 6] = colnames(input)[i]
ADF_stationary = rbind(ADF_stationary, temp)


}

ADF_stationary = ADF_stationary[, -5]
```

```
colnames(ADF_stationary) = c("ADFstat", "lag", "alternative", "pvalue", "var")

ADF_stationary = within(ADF_stationary, {
ADFstat = as.numeric(as.character(ADFstat))
lag = as.numeric(as.character(lag))
pvalue = as.numeric(as.character(pvalue))
})



### Generate Augmented Dickey-Fuller Test statistic with H1: explosive and lag
    1
for (i in 3:ncol(input)) {
if (grepl(".prev", colnames(input)[i])) {
(next)()
}
temp = adf.test(input[, i], alternative = "e", k = 1L)
temp = t(unlist(temp))
temp[, 6] = colnames(input)[i]
ADF_explosive = rbind(ADF_explosive, temp)

}

ADF_explosive = ADF_explosive[, -5]
colnames(ADF_explosive) = c("ADFstat", "lag", "alternative", "pvalue", "var")

ADF_explosive = within(ADF_explosive, {
ADFstat = as.numeric(as.character(ADFstat))
lag = as.numeric(as.character(lag))
pvalue = as.numeric(as.character(pvalue))
})

#### Kwiatkowski Phillips Schmidt Shin (KPSS) Test ####
print(paste0("KPSS lag paratmeter is set to trunc(3*sqrt(n)/13) = ", trunc(3 *
    sqrt(nrow(input))/13)))
KPSS_level = data.frame(matrix(vector(), nrow = 0, ncol = 5))
KPSS_trend = KPSS_level



### Generate KPSS Test statistic with H0: level stationarity and lag 1
for (i in 3:ncol(input)) {
```

```
if (grepl(".prev", colnames(input)[i])) {
(next)()
}
KPSS_level[i, ] = unlist(kpss.test(input[, i], null = "Level"))
KPSS_level[i, 5] = colnames(input)[i]
}

colnames(KPSS_level) = c("KPSSstat", "lag", "pvalue", "null", "var")
KPSS_level$null = gsub("KPSS_Test_for", "", KPSS_level$null)

KPSS_level = within(KPSS_level, {
KPSSstat = as.numeric(KPSSstat)
lag = as.numeric(lag)
p.value = as.numeric(pvalue)
})

### Generate KPSS Test statistic with H0: trend stationarity and lag 1
for (i in 3:ncol(input)) {
if (grepl(".prev", colnames(input)[i])) {
(next)()
}
KPSS_trend[i, ] = unlist(kpss.test(input[, i], null = "Trend"))
KPSS_trend[i, 5] = colnames(input)[i]
}

colnames(KPSS_trend) = c("KPSSstat", "lag", "pvalue", "null", "var")
KPSS_trend$null = gsub("KPSS_Test_for", "", KPSS_trend$null)

KPSS_trend = within(KPSS_trend, {
KPSSstat = as.numeric(KPSSstat)
lag = as.numeric(lag)
p.value = as.numeric(pvalue)
})
```

SPL_RAwMTSD_04_tSeriesTest_multiLag.R

```
#### Define column indices of time variables (to be excluded from tests)
timeIndices = c()

#### Register Cluster
clst = makeCluster(as.numeric(Sys.getenv("NUMBER_OF_PROCESSORS")))
registerDoParallel(clst)

timeIndices = foreach(n = 1:length(timeCols), .combine = c) %dopar% {
temp = grep(timeCols[n], colnames(input), ignore.case = TRUE)
}

notTimeCols = c(1:ncol(input))[-timeIndices]

lagLevels = c(seq(1, 21, 1))

#### Generate ADF dataset with lags from 1 to 21 #### k=22 gives an error for
    not enough data ###

ADF_multiLag_foreach = foreach(z = 1:2, .combine = rbind) %:% foreach(i =
    notTimeCols, .combine = rbind) %:% foreach(j = lagLevels,
.packages = "tseries", .combine = rbind) %dopar% {
data.frame(t(unlist(adf.test(input[, i], alternative = c("s", "e")[z], k = j)))
    , colnames(input)[i], stringsAsFactors = FALSE)
}



ADF_multiLag_foreach = ADF_multiLag_foreach[, -c(5:6)]
colnames(ADF_multiLag_foreach) = c("ADFstat", "lag", "alternative", "pvalue", "
    var")

ADF_multiLag_foreach = within(ADF_multiLag_foreach, {
ADFstat = as.numeric(as.character(ADFstat))
lag = as.numeric(as.character(lag))
pvalue = as.numeric(as.character(pvalue))
})

#### Generate KPSS dataset with tseries function ####

KPSS_multiLag_foreach = foreach(k = 1:2, .combine = rbind) %:% foreach(i =
    notTimeCols, .combine = rbind) %dopar%
```

```
{
temp = data.frame(t(unname(unlist(kpss.test(input[, i], null = c("Level", "
    Trend")[k], lshort = FALSE)))),
colnames(input)[i], stringsAsFactors = FALSE)
}
KPSS_multiLag_foreach = KPSS_multiLag_foreach[, -5]
colnames(KPSS_multiLag_foreach) = c("KPSSstat", "lag", "pvalue", "null", "var")
KPSS_multiLag_foreach$null = gsub("KPSS Test for", "", KPSS_multiLag_foreach$
    null)

#### Generate KPSS dataset with customized tseries function #### Changed
    returned object from list to vector
#### Removed automatic choice of lag-parameter
kpss.test.lag = function(x, null = c("Level", "Trend"), k = 1) {
if ((NCOL(x) > 1) || is.data.frame(x))
stop("x is not a vector or univariate time series")
DNAME = deparse(substitute(x))
null = match.arg(null)
x = as.vector(x, mode = "double")
n = length(x)
if (null == "Trend") {
t = 1:n
e = residuals(lm(x ~ t))
table = c(0.216, 0.176, 0.146, 0.119)
} else if (null == "Level") {
e = residuals(lm(x ~ 1))
table = c(0.739, 0.574, 0.463, 0.347)
}
tablep = c(0.01, 0.025, 0.05, 0.1)
s = cumsum(e)
eta = sum(s^2)/(n^2)
s2 = sum(e^2)/n
l = k

s2 = .C("R_pp_sum", as.vector(e, mode = "double"), as.integer(n), as.integer(l)
    , s2 = as.double(s2), PACKAGE = "tseries")$s2
STAT = eta/s2
PVAL = approx(table, tablep, STAT, rule = 2)$y
if (is.na(approx(table, tablep, STAT, rule = 1)$y))
if (PVAL == min(tablep))
```

```
warning("p-value smaller than printed p-value") else warning("p-value greater
    than printed p-value")
data.frame(KPSSstat = STAT, lag = l, pvalue = PVAL, null = paste(null, "
    Stationarity"), var = DNAME, stringsAsFactors = FALSE)
}

KPSS_v2_multiLag_foreach = foreach(k = 1:2, .combine = rbind) %:% foreach(i =
    notTimeCols, .combine = rbind) %:%
foreach(j = lagLevels, .combine = rbind) %dopar% {
data.frame(kpss.test.lag(input[, i], null = c("Level", "Trend")[k], k = j),
    colnames(input)[i], stringsAsFactors = FALSE)
}

KPSS_v2_multiLag_foreach = KPSS_v2_multiLag_foreach[, -5]
colnames(KPSS_v2_multiLag_foreach) = c("KPSSstat", "lag", "pvalue", "null", "
    var")
stopCluster(clst)
```

SPL_RAwMTSD_05_tSeries_lm.R

```
reg.now = lm(Gini.Coefficient.now ~ GDP.now + Inflation.now + Unemployment.now,
     input)
summary(reg.now)
str(summary(reg.now))
plot(reg.now$residuals)

reg.fd = lm(Gini.Coefficient.fd ~ GDP.fd + Inflation.fd + Unemployment.fd,
    input)
summary(reg.fd)
str(summary(reg.fd))
plot(reg.fd$residuals)

kpss.test(reg.now$residuals, null = "L")
kpss.test(reg.fd$residuals, null = "L")

reg.now.logit = glm(Gini.Coefficient.now ~ GDP.now + Inflation.now +
    Unemployment.now, input, family = binomial(link = "logit"))
summary(reg.now.logit)
str(summary(reg.now.logit))
plot(reg.now.logit$residuals)
```

SPL_RAwMTSD_06_funct_benchmark.R

```
#### Find codes to be benchmarked ####
cd = getwd()
cd.new = paste0(cd, "/benchm")
setwd(cd.new)

#### Setting number of code runs ####
t = 20

#### Test base::for
time_test_funct = summary(microbenchmark(source("benchm_for.R"), times = t,
    unit = "s"))

#### Test foreach::foreach with maximum number of processor cores ####
clst = makeCluster(as.numeric(Sys.getenv("NUMBER_OF_PROCESSORS")))
registerDoParallel(clst)

time_test_funct = rbind(time_test_funct, summary(microbenchmark(source("benchm_
    foreach.R"), times = t, unit = "s")))
time_test_funct$expr = gsub("foreach.R", paste0("foreach_", as.numeric(Sys.
    getenv("NUMBER_OF_PROCESSORS")), "CPUs.R"),
time_test_funct$expr)
stopCluster(clst)

#### Test base::sapply and base::lapply ####
time_test_funct = rbind(time_test_funct, summary(microbenchmark(source("benchm_
    apply.R"), times = t, unit = "s")))

#### Test foreach::foreach with 1 processor core ####
clst = makeCluster(1)
registerDoParallel(clst)

time_test_funct = rbind(time_test_funct, summary(microbenchmark(source("benchm_
    foreach.R"), times = t, unit = "s")))
time_test_funct$expr = gsub("foreach.R", "foreach_1CPU.R", time_test_funct$expr
    )
stopCluster(clst)

#### Format output data.frame ####
time_test_funct$expr = gsub("source(\"benchm_","",time_test_funct$expr,fixed
    =TRUE)
```

```
time_test_funct$expr = gsub(".R\")", "", time_test_funct$expr, fixed = TRUE)
time_test_funct = time_test_funct[order(time_test_funct$mean), ]
rownames(time_test_funct) = c(1:nrow(time_test_funct))

#### Return to project working directory ####
setwd(cd)
rm(cd.new, cd)
```

SPL_RAwMTSD_07_Export_tex.R

```
#### Set export directory ####
cd = getwd()
cd.new = paste0(cd, "/tex")
if (!dir.exists(paste0(getwd(), "/tex"))) {
print("Creating folder '/tex'")
dir.create(cd.new)
} else {
print("Folder '/tex' exists. Careful to not overwrite existing text and tex
    files")
}

setwd(cd.new)

partDF = cbind(DF_base[DF_base$test.level == "UR.drift.tTrend", c(1, 5)], ADF_
    stationary[, c(4, 5)])
partDF$var = as.character(partDF$var)

if (identical(partDF[, 1], partDF[, 4])) {
print("Variables match")
partDF = partDF[, -4]
} else {
print("Variable names do not match")
}

colnames(partDF) = c("varName", "DF.p-value", "ADF.p-value")

partDF2 = cbind(DF_base[DF_base$test.level == "UR.drift.tTrend", c(1, 3)], ADF_
    stationary[, c(1, 5)])
partDF2$var = as.character(partDF2$var)

if (identical(partDF2[, 1], partDF2[, 4])) {
```

```
print("Variables match")
partDF2 = partDF2[, -4]
} else {
print("Variable names do not match")
}

colnames(partDF2) = c("varName", "DF.t-value", "ADF.t-value")

stargazer(partDF, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    pval_comp.txt")
stargazer(partDF, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    pval_comp.tex")

stargazer(partDF2, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    tval_comp.txt")
stargazer(partDF2, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    tval_comp.tex")

stargazer(reg.now, reg.fd, align = TRUE, out = "reg_table.txt")
stargazer(reg.now, reg.fd, align = TRUE, out = "reg_table.tex")

stargazer(time_test_funct, align = TRUE, summary = FALSE, rownames = FALSE, out
     = "benchm_results.txt")
stargazer(time_test_funct, align = TRUE, summary = FALSE, rownames = FALSE, out
     = "benchm_results.tex")

stargazer(DF_base, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    pval_comp.txt")
stargazer(DF_base, summary = FALSE, align = TRUE, rownames = FALSE, out = "df_
    pval_comp.tex")

multiLagsADF = ADF_multiLag[ADF_multiLag$var %in% c("GDP.now", "Gini.
    Coefficient.now") & ADF_multiLag$alternative ==
"stationary", ]
multiLagsKPSS = KPSS_multiLag_v2[KPSS_multiLag_v2$var %in% c("GDP.now", "Gini.
    Coefficient.now"), ]

export.envir = c("ADF_explosive", "ADF_stationary", "KPSS_level", "KPSS_trend",
     "multiLagsADF", "multiLagsKPSS")

for (k in 1:length(export.envir)) {
```

```
arrangeCols = c("var", "lag", "pvalue", ifelse(grepl("ADF", export.envir[k]), "
    ADFstat", "KPSSstat"), ifelse(grepl("ADF",
export.envir[k]), "alternative", "null"))
temp = eval(parse(text = export.envir[k]), envir = .GlobalEnv)
temp = temp[, arrangeCols]
stargazer(temp, align = TRUE, summary = FALSE, rownames = FALSE, out = paste0(
    export.envir[k], ".txt"))
stargazer(temp, align = TRUE, summary = FALSE, rownames = FALSE, out = paste0(
    export.envir[k], ".tex"))


}
#### Return to project working directory ####
setwd(cd)
rm(cd.new, cd)
```