# Macroeconomic Forecasting - Forecasting the US Savings Rate

*Anders Christiansen*

*December 13, 2015*

This R code is based on the Final Assignment for the IMF course "Macroeconomic Forecasting" (https://courses.edx.org/courses/course-v1:IMFx+MFx+2015T3/info). The course is conducted in E-Views and this script aims to replicate the results in R. Many of the results have slight discrepencies or are given using a different type of test.

If you have any suggestions for how to make the results more consistent with the Final Project, please let me know by contacting me at: http://aachristiansen.com/contact/

You can download the R script at http://aachristiansen.com/ForecastingUSSavings.R

I hope this helps you begin to apply what you learned in E-Views in R!

## Set-Up

First let us load the packages and data we will be working with:

```
require(vars)
require(tsDyn)
data = read.csv("http://www.aachristiansen.com/usa_cy.csv")
```

Convert data frame into time series vectors

```
for(i in 1:dim(data)[2])
  assign(names(data)[i],window(ts(data[,i],
                                  frequency = 4,
                                  start = c(1945,1),
                                  end=c(2015, 4)),
         start = c(1961,1),
         end = c(2007,4)))
```

Create time series for the Test data (2008-2014)

```
for(i in 1:dim(data)[2])
  assign(paste0(names(data)[i],"Test"),window(ts(data[,i],
                                  frequency = 4,
                                  start = c(1945,1)),
                                start = c(2008,1),
                                end = c(2014,3)))
```

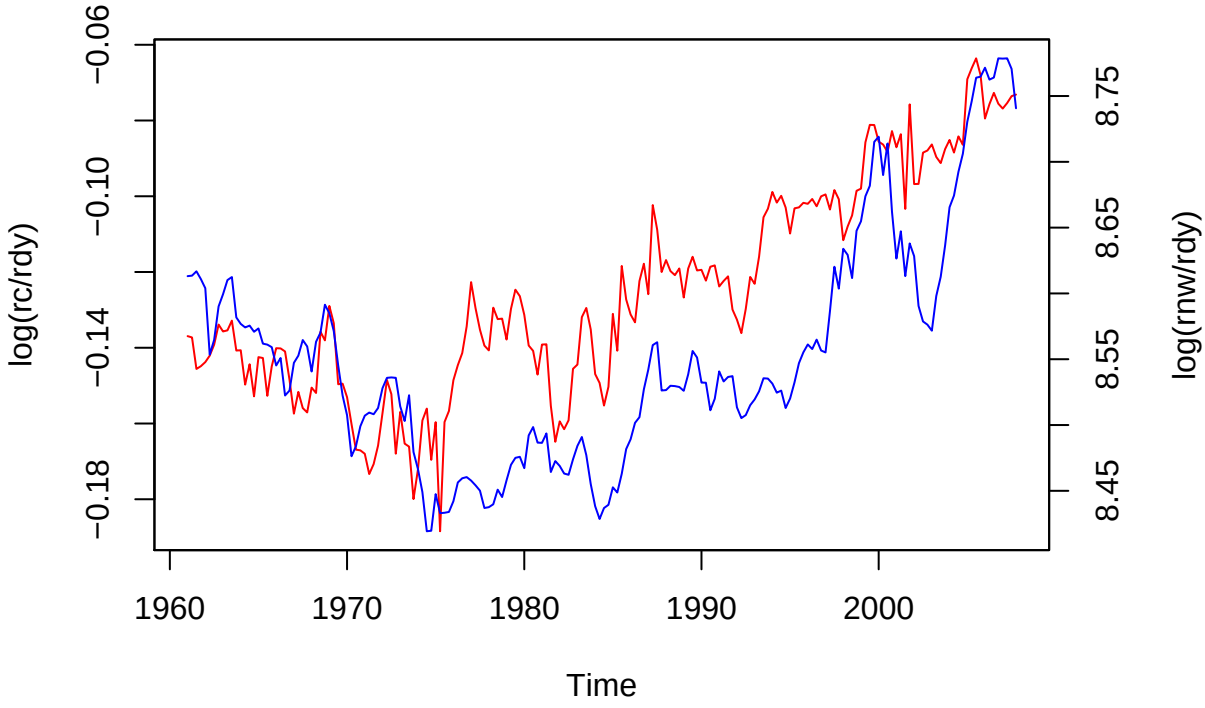
Calculate the Savings Rate (Income minus Consumption minus Government Transfers minus Interest Payments as a percentage of Income)

```
saving_rate = ts(
  100*(rdy - (rc + ((gov_transfers + interest_payments/y_deflator))))/rdy,
               frequency = 4, start = c(1961,1))
saving_rate_test = ts(
  100*(rdyTest - (rcTest +
               ((gov_transfersTest + interest_paymentsTest/y_deflatorTest))))/rdyTest,
                     frequency = 4, start = c(2008,1))
```

## Question 8.1

Plot the log of Real Consumption (rc) divided by Real Disposable Income (rdy) next to the log of Real Net Worth (rnw) divided by Real Disposable Income (rdy). (Watch out how you interpret the graph! The axes are different than in E-Views)

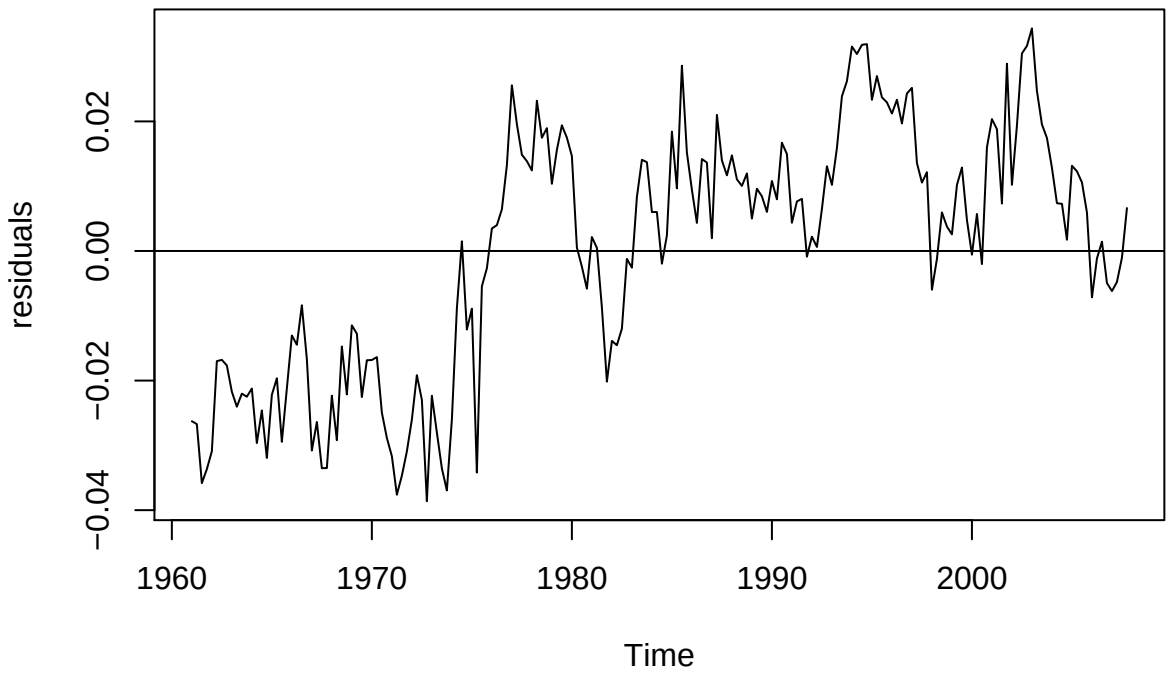
```
par(mar=c(5,4,4,5)+.1)
plot.ts(log(rc/rdy),
        type="l",
        col="red")
par(new=TRUE)
plot.ts(log(rnw/rdy),
        type="l",
        col="blue",
        xaxt="n", yaxt="n",xlab="",ylab="")
axis(4)
mtext("log(rnw/rdy)",side=4,line=3)
```




## Question 8.2

Plot the residuals of a simple regression of the log of Real Consumption on the log of Real Net Worth to see how they change over time.

```
fit = lm(log(rc/rdy) ~ log(rnw/rdy))
residuals = ts(fit$residuals,frequency = 4, start = c(1961,1))
plot.ts(residuals)
abline(h=0)
```




## Question 8.3

Set up a dummy variable to represent the structural break.

```
n = length(dateid01)
zeros = (1975-1961)*4 + 3
sb_1975_4 = ts(c(rep(0,zeros), rep(1, n-zeros)),frequency = 4, start = c(1961,1))
```

## Question 8.4 - 8.5

Test for unit roots of Real Consumption, Real Disposable Income, and Real Net Worth. Tests for stationarity using the Augmented Dickey-Fuller Test. To automatically select the number of lags add the argument:

selectlags = c("Fixed", "AIC", "BIC")

(Results vary slightly from EViews and output has been shortened)

```
summary(ur.df(y = log(rc), type = "drift", lags = 2))

## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.019282   0.008935   2.158   0.0322 *
## z.lag.1     -0.001618   0.001038  -1.560   0.1206
```

```
## z.diff.lag1  0.189095   0.072915   2.593   0.0103 *
## z.diff.lag2  0.179019   0.072849   2.457   0.0149 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.006397 on 181 degrees of freedom
## Multiple R-squared:  0.1062, Adjusted R-squared:  0.09136
## F-statistic: 7.167 on 3 and 181 DF,  p-value: 0.0001421
```

```
summary(ur.df(y = log(rdy), type = "drift", lags = 0))
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03734    0.01165   3.205  0.00159 **
## z.lag.1     -0.00336    0.00136  -2.471  0.01440 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.008232 on 185 degrees of freedom
## Multiple R-squared:  0.03194,    Adjusted R-squared:  0.02671
## F-statistic: 6.104 on 1 and 185 DF,  p-value: 0.0144
```

```
summary(ur.df(y = log(rnw), type = "drift", lags = 4))
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.011916   0.042609  -0.280 0.780060
## z.lag.1      0.001134   0.002501   0.453 0.650781
## z.diff.lag1  0.196248   0.073627   2.665 0.008400 **
## z.diff.lag2  0.057156   0.073683   0.776 0.438959
## z.diff.lag3  0.199181   0.073698   2.703 0.007548 **
## z.diff.lag4 -0.267532   0.073932  -3.619 0.000386 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01583 on 177 degrees of freedom
## Multiple R-squared:  0.1264, Adjusted R-squared:  0.1018
## F-statistic: 5.124 on 5 and 177 DF,  p-value: 0.0002053
```

# QUESTION 8.6

Select the correct number of lags for the Vector Auto-Regressive model given a variety of selection criteria. (Results vary slightly from EViews)

```
VARselect(cbind(log(rc),log(rdy),log(rnw)),
          type="const",
          exogen = data.frame(sb_1975_4 = sb_1975_4))$selection
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      5      2      1      5
```

# QUESTION 8.7 - 8.9

Set up the VAR model and run a series of tests for autocorrelation, heteroskedasticity, and normality. (The tests in the VAR package are different from E-Views)

```
SimpleVar = VAR(cbind(log(rc),log(rdy),log(rnw)),
             p = 2,
             type="const",
             exogen = data.frame(sb_1975_4 = sb_1975_4))

serial.test(SimpleVar)
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object SimpleVar
## Chi-squared = 155.14, df = 126, p-value = 0.0399
```

```
arch.test(SimpleVar,4)
```

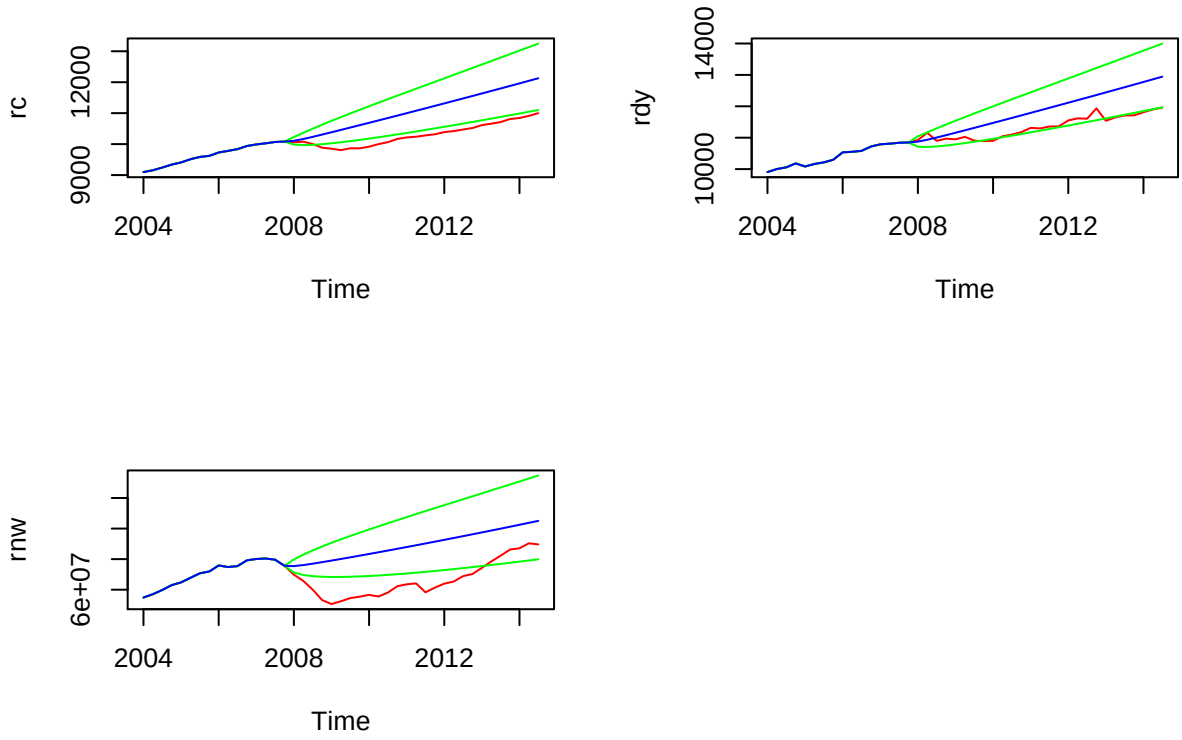
```
##
##  ARCH (multivariate)
##
## data:  Residuals of VAR object SimpleVar
## Chi-squared = 241.5, df = 180, p-value = 0.001509
```

```
normality.test(SimpleVar)
```

```
## $JB
##
##  JB-Test (multivariate)
##
## data:  Residuals of VAR object SimpleVar
## Chi-squared = 100.98, df = 6, p-value < 2.2e-16
##
##
## $Skewness
##
##  Skewness only (multivariate)
##
## data:  Residuals of VAR object SimpleVar
## Chi-squared = 25.047, df = 3, p-value = 1.51e-05
##
##
## $Kurtosis
##
##  Kurtosis only (multivariate)
##
## data:  Residuals of VAR object SimpleVar
## Chi-squared = 75.935, df = 3, p-value = 2.22e-16
```

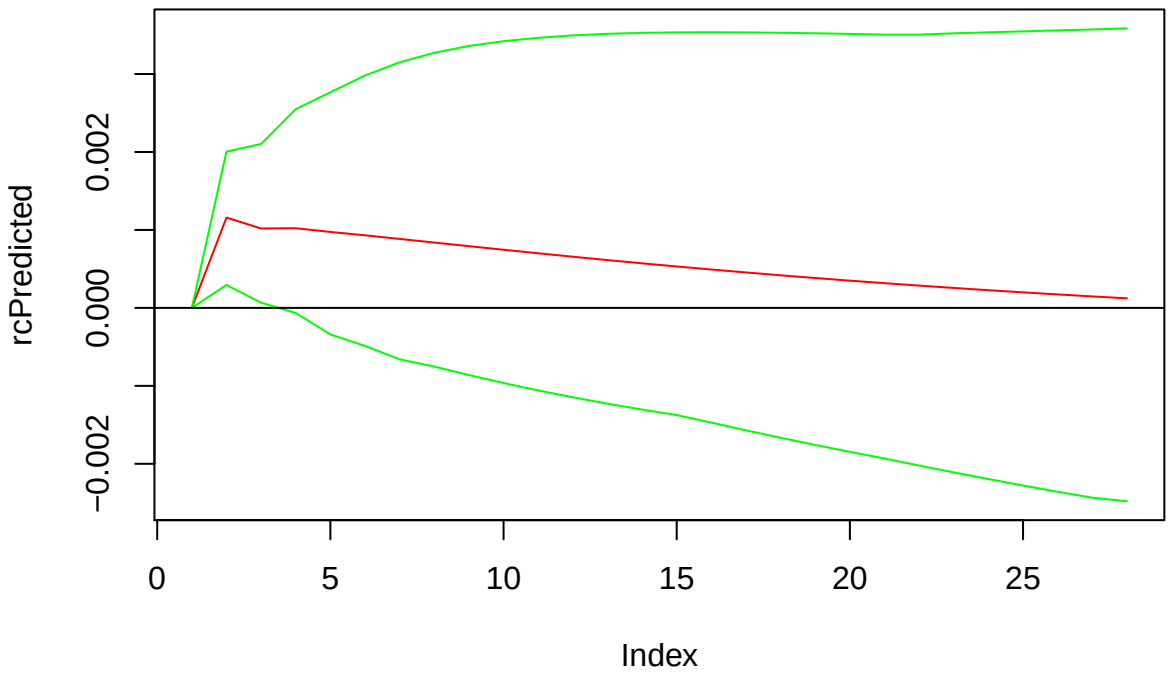
Forecasting variables with VAR model and 95% confidence intervals.

```
VARForecast = predict(SimpleVar,n.ahead = 27,dumvar=data.frame(sb_1975_4 = rep(1,27)), ci = .95)
```




Impulse Response of Real Consumption to a shock in Real Disposable Income.

```
VARImpulse = irf(SimpleVar, impulse = "log.rdy.", n.ahead = 27, ci = .95)
```




## QUESTION 8.10

In order to determine the number of cointegrating vectors to use when estimating the Vector Error Correcting Model (VECM), we must run the Johansen Cointegration Test. (Result is different from EViews and results are truncated)

```
VECMTest = ca.jo(cbind(log(rc),log(rdy),log(rnw)),
                 type="eigen", ecdet = "const", K = 2,
                 dumvar = data.frame(sb_1975_4 = sb_1975_4))
summary(VECMTest)

## Values of teststatistic and critical values of test:
##
##           test 10pct  5pct  1pct
## r <= 2 |  2.00  7.52  9.24 12.97
## r <= 1 | 23.58 13.75 15.67 20.20
## r = 0  | 51.56 19.77 22.00 26.81
```

## QUESTION 8.11

Test the change in Unemployment (unemp) and the log of Consumer Sentiment (consumer_sentiment) for stationarity using the Augmented Dickey-Fuller Test. (Results are truncated.)

```
summary(ur.df(y = diff(unemp), type = "drift", selectlags = "AIC"))

## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.00274    0.02178  -0.126    0.900
## z.lag.1     -0.46313    0.07573  -6.116 5.73e-09 ***
## z.diff.lag  -0.12048    0.07367  -1.635    0.104
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.296 on 182 degrees of freedom
## Multiple R-squared:  0.2734, Adjusted R-squared:  0.2654
## F-statistic: 34.23 on 2 and 182 DF,  p-value: 2.399e-13

summary(ur.df(y = log(consumer_sentiment), type = "drift", selectlags = "AIC"))

## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.48222    0.16940   2.847  0.00492 **
## z.lag.1     -0.10824    0.03791  -2.855  0.00480 **
## z.diff.lag  -0.19171    0.07296  -2.627  0.00933 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07098 on 183 degrees of freedom
## Multiple R-squared:  0.09993,    Adjusted R-squared:  0.0901
## F-statistic: 10.16 on 2 and 183 DF,  p-value: 6.548e-05
```

# QUESTION 8.12

Using the Johansen Cointegration Test object, the urca package can construct the VECM model.

```
difUnem = window(diff(ts(data$unemp, frequency = 4, start = c(1945,1))),
                 start = c(1961,1),
                 end = c(2007,4))

VECMurca = ca.jo(cbind(log(rc),log(rdy),log(rnw)),
                 type="eigen", ecdet = "const", K = 2,
                 dumvar = data.frame(sb_1975_4 = sb_1975_4,
                                     difUnem = difUnem,
                                     consConf = log(consumer_sentiment)),
                 spec="longrun")

cajorls(VECMurca, 1)


## $rlm
##
## Call:
## lm(formula = substitute(form1), data = data.mat)
##
## Coefficients:
```

```
##               log.rc..d  log.rdy..d  log.rnw..d
## ect1          -0.187449   0.010152   -0.126953
## sb_1975_4      0.001552  -0.004008    0.003795
## difUnem       -0.007784  -0.006280   -0.001941
## consConf       0.018493   0.001475    0.012629
## log.rc..dl1   -0.297075   0.205715    0.170085
## log.rdy..dl1   0.182107  -0.197673   -0.289025
## log.rnw..dl1   0.065724   0.059474    0.169091
##
##
## $beta
##                    ect1
## log.rc..l2   1.0000000
## log.rdy..l2 -0.9092629
## log.rnw..l2 -0.1099968
## constant     1.6277940
```

## Dynamic Forecast

In order to produce a dynamic forecast the VECM function of the "tsDyn" package is used.

```
SimpleVECM = VECM(cbind(log(rc),log(rdy),log(rnw)), lag=1, r = 1, include = "const",
     estim ="ML", LRinclude = "const",
     exogen = data.frame(sb_1975_4 = sb_1975_4))

VECMDynForecast = predict(SimpleVECM,n.ahead = 27,exoPred=data.frame(sb_1975_4 = rep(1,27)))

## Calculate the Savings Rate using the dynamic forecast.
saving_rate_forecastD = ts(
  100*(exp(VECMDynForecast[,"log(rdy)"]) - (exp(VECMDynForecast[,"log(rc)"]) +
  ((gov_transfersTest + interest_paymentsTest/y_deflatorTest))))
  /exp(VECMDynForecast[,"log(rdy)"]),
                       frequency = 4, start = c(2008,1))
```

## Static Forecast

For the static forecast we predict the Savings Rate only one period ahead. The for loop adds the new data for the quarter onto the existing data to predict one quarter ahead.

```
VECMStaticForecast = VECMDynForecast

rc2=rc;rdy2=rdy;rnw2=rnw;sb_1975_4_2=sb_1975_4;
for(i in 1:dim(VECMDynForecast)[1]) {
  rc2 = ts(c(rc2,rcTest[i]), frequency = 4, start = c(1961,1))
  rdy2 = ts(c(rdy2,rdyTest[i]), frequency = 4, start = c(1961,1))
  rnw2 = ts(c(rnw2,rnwTest[i]), frequency = 4, start = c(1961,1))
  sb_1975_4_2 = ts(c(sb_1975_4_2,1), frequency = 4, start = c(1961,1))
  SimpleVECM = VECM(cbind(log(rc2),log(rdy2),log(rnw2)), lag=1, r = 1,
       include = "const", estim ="ML", LRinclude = "const",
       exogen = data.frame(sb_1975_4_2 = sb_1975_4_2))
```

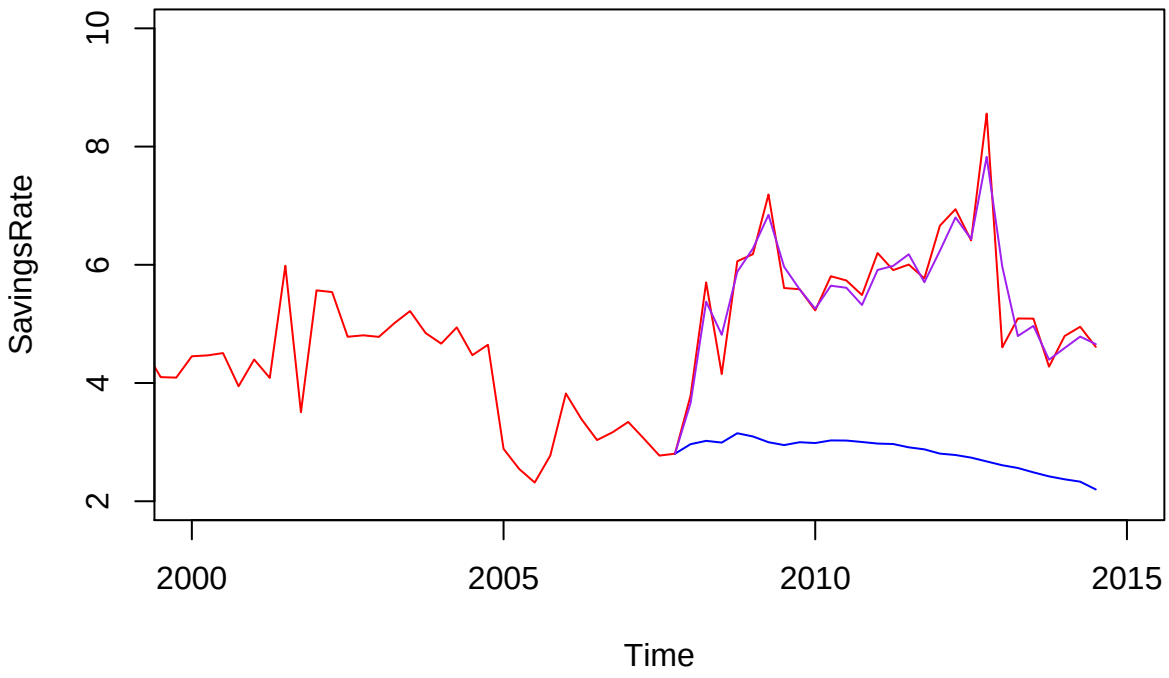
```
  VECMStaticForecast[i,] = predict(SimpleVECM,n.ahead = 1,
       exoPred=data.frame(sb_1975_4_2=1))
}

saving_rate_forecastS = ts(
  100*(exp(VECMStaticForecast[,"log(rdy)"]) - (exp(VECMStaticForecast[,"log(rc)"])
  + ((gov_transfersTest + interest_paymentsTest/y_deflatorTest))))
  /exp(VECMStaticForecast[,"log(rdy)"]),
                           frequency = 4, start = c(2008,1))
```

## Comparison

```
SavingsRate = ts(
  c(saving_rate,saving_rate_test), frequency = 4, start = c(1961,1))
DynamicForecast = ts(
  c(saving_rate[length(saving_rate)],saving_rate_forecastD), frequency = 4, start = c(2007,4))
StaticForecast = ts(
  c(saving_rate[length(saving_rate)],saving_rate_forecastS), frequency = 4, start = c(2007,4))

plot(SavingsRate, col="red",
     ylim = c(2,10),
     xlim = c(2000,2015))
lines(DynamicForecast, col="blue")
lines(StaticForecast, col="purple")
```




## Calculating performance vs baseline

```
rmse <- function(error) sqrt(mean(error^2))

## Static Forecast RMSE
rmse(saving_rate_forecastS - saving_rate_test)
```

```
## [1] 0.3767508
```

```
## Baseline using last quarters Savings Rate
rmse(c(saving_rate[length(saving_rate)],
       saving_rate_test[1:(length(saving_rate_test)-1)]) - saving_rate_test)
```

```
## [1] 1.187646
```

```
## Dynamic Forecast RMSE
rmse(saving_rate_forecastD - saving_rate_test)
```

```
## [1] 2.988718
```

```
## Baseline 2007 Q1
rmse(saving_rate[length(saving_rate)] - saving_rate_test)
```

```
## [1] 3.010411
```

```
## T Statistic Static Forecast
errorDifStatic = (saving_rate_forecastS - saving_rate_test)^2 -
     (c(saving_rate[length(saving_rate)],
        saving_rate_test[1:(length(saving_rate_test)-1)]) - saving_rate_test)^2

t.test(errorDifStatic,mu=0)
```

```
##
##  One Sample t-test
##
## data:  errorDifStatic
## t = -2.3721, df = 26, p-value = 0.02537
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -2.367824 -0.169298
## sample estimates:
## mean of x
## -1.268561
```

```
## T Statistic Dynamic Forecast

errorDifDynamic = (saving_rate_forecastD - saving_rate_test)^2 -
     (saving_rate[length(saving_rate)] - saving_rate_test)^2

t.test(errorDifDynamic,mu=0)
```

```
##
##  One Sample t-test
##
## data:  errorDifDynamic
## t = -0.5106, df = 26, p-value = 0.6139
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.6540505  0.3937690
## sample estimates:
##  mean of x
## -0.1301408
```

## Conclusion

The VECM model is significantly more effective than the baseline at short term predictions, however the long-term forecast did not do significantly better than the baseline at predicting the Great Recession.

It also appears that the fundamental relationship did not change between disposable income, net worth, and consumption. Instead, following 2008 a shock to real net worth and disposable income was the culprit that caused a forecast outside the 95% confidence interval.