Forecasting Task - Chapter 1

Bernardo Calvente e Ricardo Castro

2024-08-15

1 Setup

The following packages were used:

```r
library(tidyverse)
library(patchwork)
library(glue)
library(stargazer)
library(tseries)
library(lmtest)
library(sandwich)
library(strucchange)

theme_set(theme_bw())
```

Also, the function below was created to print the output of the ADF test.

```r
output_dftest <- function(data,
  nlag = NULL, pval = TRUE, index = TRUE, ...) {
  aTSA::adf.test(data, nlag = nlag, output = FALSE) %>%
    imap_dfr(~ tibble(type = .y, as_tibble(.x))) %>%
    mutate(
      p.value = if (pval) {glue("({round(p.value, 2)})")} else {""},
      ADF = round(ADF, 2)
    ) %>%
    unite(Statistic, ADF, p.value, sep = " ") %>%
    pivot_wider(names_from = type, values_from = "Statistic") %>%
    set_names("Lag", glue("Type {1:3}")[index]) %>%
    stargazer(summary = FALSE, header = FALSE, table.placement = "H")
}
```

Loading the data:

```r
data <- read_csv("chap1_g5.csv")
```

From the true model, the relation between Y and X is as follows:

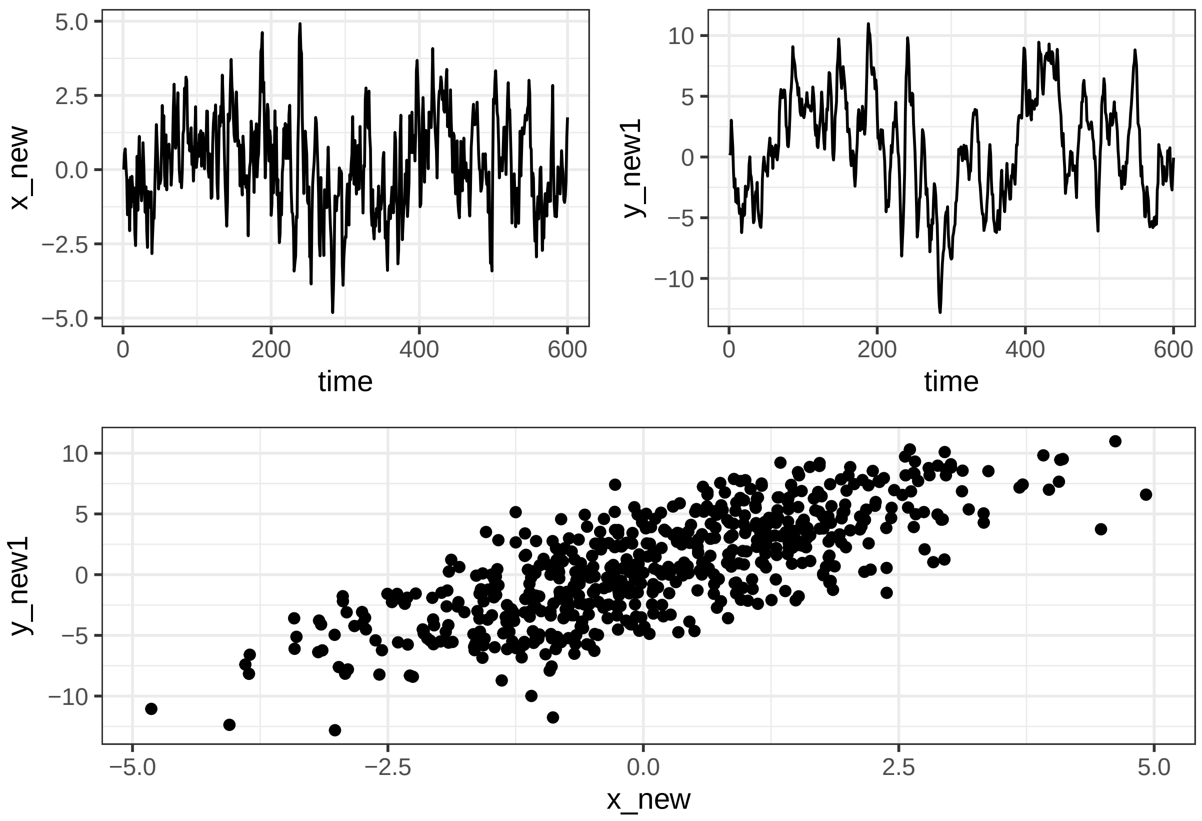
$$y(t) = x(t) + 0.8y(t-1) - 0.2x(t-1) + e_1(t) - e_2(t)$$

2 Part 1

2.1 Q1. Stationarity Analisys

Plots of the series' historic values:

```r
gx <- ggplot(data, aes(time, x_new)) +
  geom_line()

gy <- ggplot(data, aes(time, y_new1)) +
  geom_line()

gxy <- ggplot(data, aes(x_new, y_new1)) +
  geom_point()

gx + gy + gxy + plot_layout(design = "12\n33")
```



Generally, both series appear to be stationary, varying in a fixed window - -5 to 5 for X, and -10 to 10 for Y.

In the last plot, we can also see that the series have a positive correlation between them.

To get further evidence on the stationarity of the series, we can run the Augmented Dickey-Fuller test. As the series don't seem to present drift nor tendency, we focus on the "Type 1" version of the test, but all are presented below. Aditionally, we considered the test with 0, 1, and 2 lags.

ADF tests on X and Y can be seen below. We can see that the null Hypothesis is always rejected, and we have further evidence that both series are stationary.

```r
output_dftest(ts(data$x_new), nlag = 3)
```

Table 1:

	Lag	Type 1	Type 2	Type 3
1	0	-8.34 (0.01)	-8.4 (0.01)	-8.39 (0.01)
2	1	-8.57 (0.01)	-8.63 (0.01)	-8.63 (0.01)
3	2	-7.98 (0.01)	-8.04 (0.01)	-8.04 (0.01)

```r
output_dftest(ts(data$y_new1), nlag = 3)
```

Table 2:

	Lag	Type 1	Type 2	Type 3
1	0	-3.34 (0.01)	-3.39 (0.01)	-3.39 (0.06)
2	1	-4.83 (0.01)	-4.91 (0.01)	-4.91 (0.01)
3	2	-5.39 (0.01)	-5.47 (0.01)	-5.46 (0.01)

2.2 Q2-4. OLS Comparison

Now, we are going to regress Y on X, using both standard and recursive OLS, and compare the results.

2.2.1 Standard OLS

```r
n <- nrow(data)

mod_std <- lm(y_new1 ~ x_new, data[101:n,])

# Predictions:
pred_std <- predict(mod_std, data[(n - 199):n,], se.fit = TRUE)[1:2] %>%
  bind_cols() %>%
  mutate(mod = "Standard OLS", time = (n - 199):n)

# Coefficient:
stargazer(mod_std, header = FALSE, table.placement = "H")
```

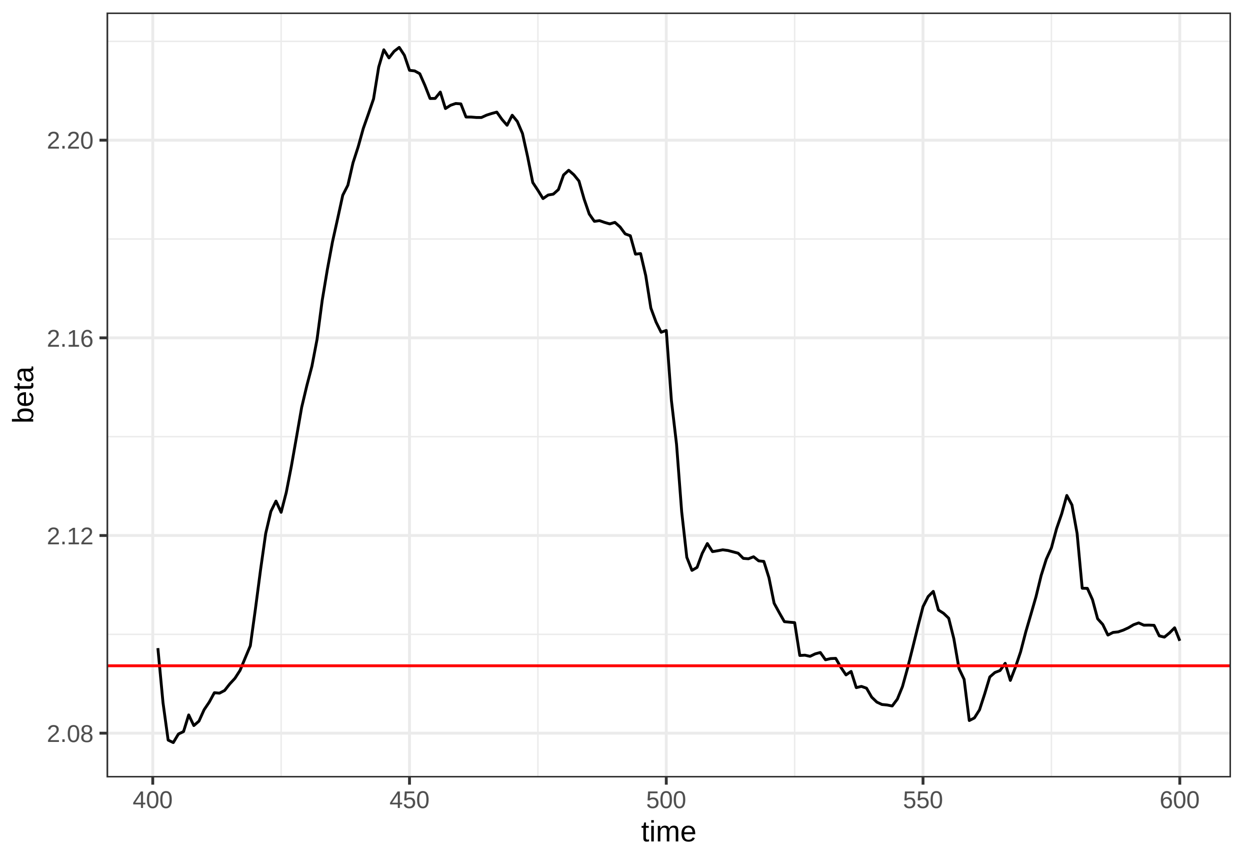
Table 3:

	Dependent variable:
	y_new1
x_new	2.094***
	(0.079)
Constant	0.513***
	(0.130)
Observations	500
R^2	0.585
Adjusted R^2	0.584
Residual Std. Error	2.886 (df = 498)
F Statistic	701.147*** (df = 1; 498)
Note:	*p<0.1; **p<0.05; ***p<0.01

We find that the multiplier related to X is approximately 2.093.

2.2.2 Recursive OLS

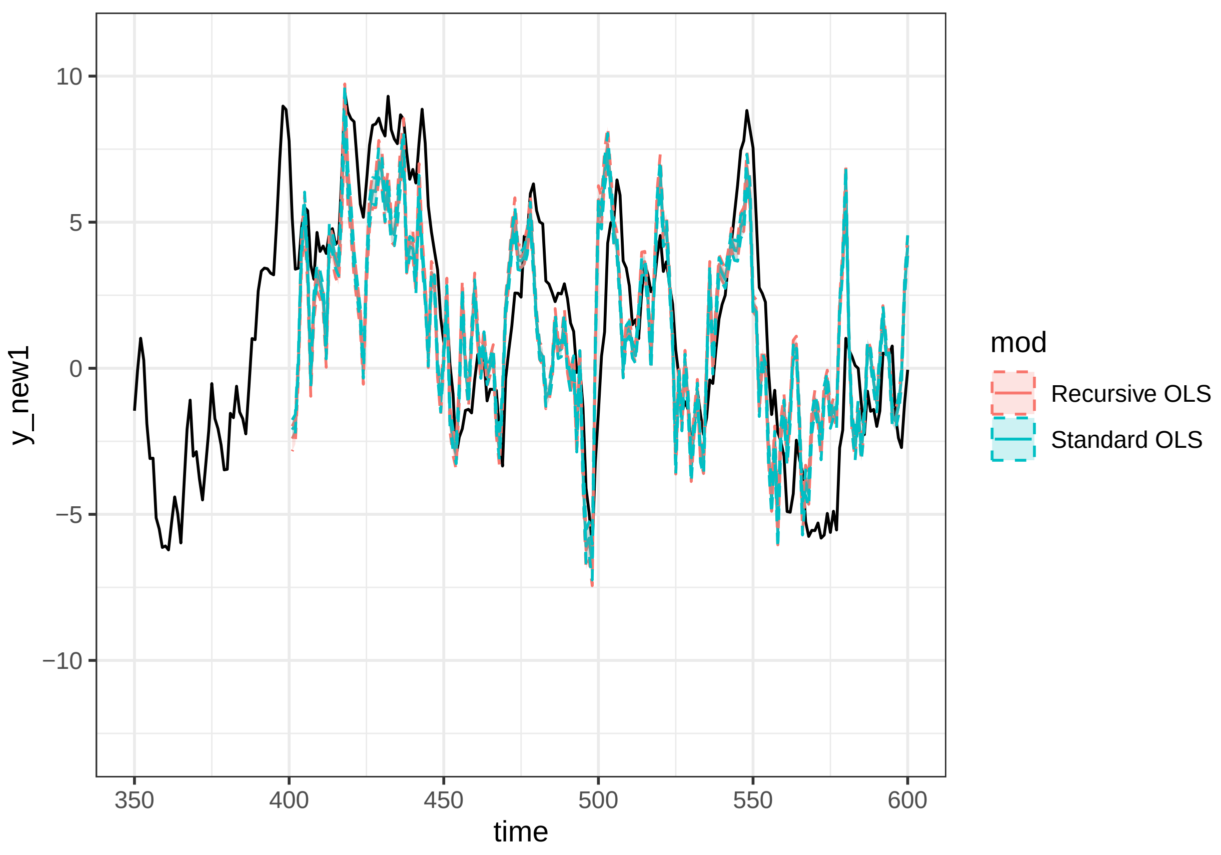
```r
mod_rec <- list()
pred_rec <- list()

for (i in 1:200) {
  mod_rec[[i]] <- lm(y_new1 ~ x_new, data[101:(n - 201 + i),])
  pred_rec[[i]] <- predict(mod_rec[[i]], data[n - 201 + i + 1,], se.fit = TRUE)[1:2]
}

# Predictions:
pred_rec <- pred_rec %>%
  bind_rows() %>%
  mutate(mod = "Recursive OLS", time = (n - 199):n)

# Coefficients:
tibble(
  beta = map_dbl(mod_rec, ~ .x$coefficients["x_new"]),
  time = (n - 199):n
) %>%
  ggplot(aes(time, beta)) +
  geom_line() +
  geom_hline(yintercept = mod_std$coefficients["x_new"], color = "red")
```



In the graph above we can see the comparison with the standard OLS coefficient (in red). We can see that, with this method, the X coefficient varies over time, but not necessarily a lot, differing at most, 0.11 from the standard model's beta.

2.2.3 Comparing the Predictions

```r
pred <- bind_rows(pred_std, pred_rec)

ggplot(data, aes(time)) +
  geom_line(aes(y = y_new1)) +
  geom_line(aes(y = fit, color = mod), pred) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit, ymax = fit + 1.96*se.fit, color = mod, fill = mod),
    pred, linetype = 2, alpha = 0.2
  ) +
  xlim(350, 600)
```



We can see that both models are very similar. We can compare them further using some descriptive statistics:

```
map_dfr(c("Standard OLS", "Recursive OLS"), function(x) {
  error <- filter(data, time >= 401)$y_new1 - filter(pred, mod == x)$fit
  list(
    Model = x,
    RMSE = (sqrt(sum(error^2)) / 200) %>% round(5),
    MAE = (sum(abs(error)) / 200) %>% round(5)
  )
}) %>%
  stargazer(summary = FALSE, header = FALSE, table.placement = "H")
```

Table 4:

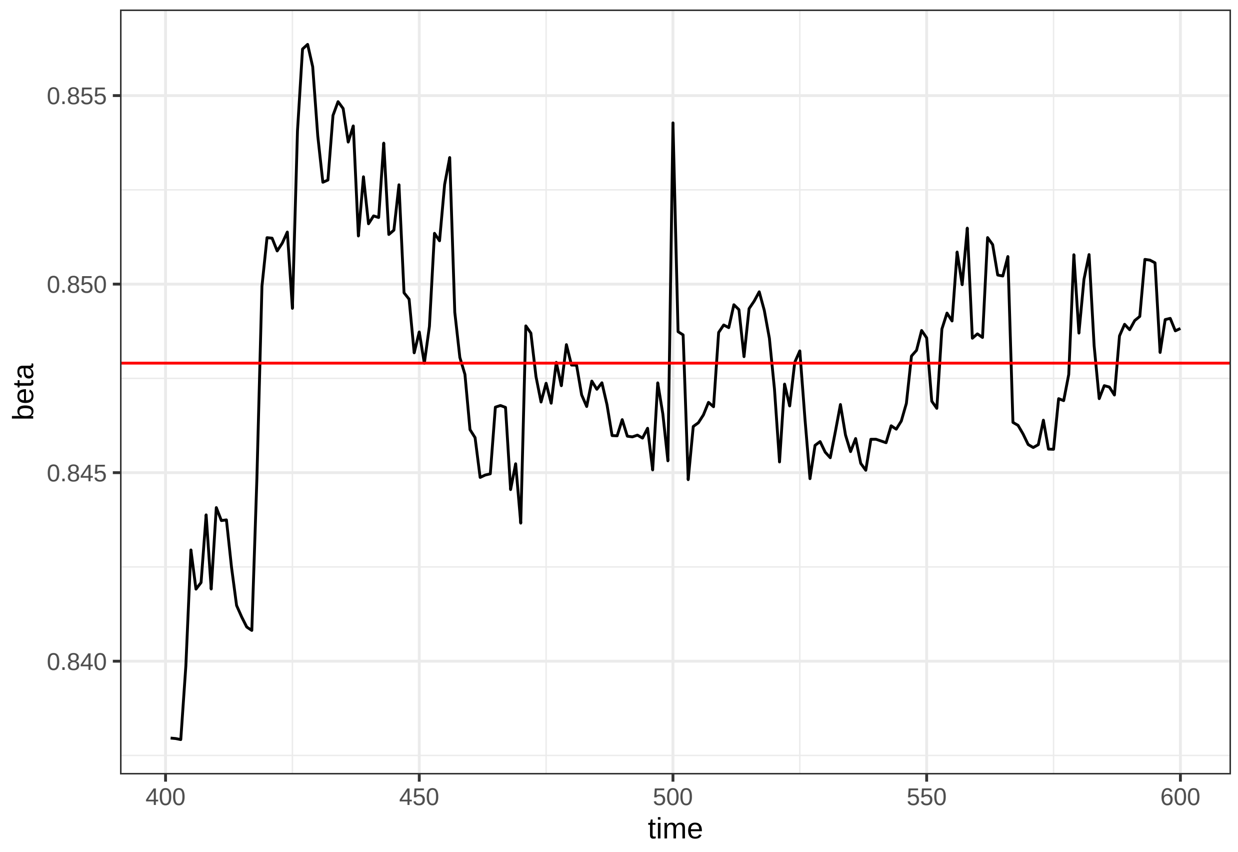
	Model	RMSE	MAE
1	Standard OLS	0.18727	2.15805
2	Recursive OLS	0.18951	2.18684

Again, further evidence that both models yielded indeed very similar results. This is probably related to the fact that the recursive OLS model is useful to capture the dynamic of the process, but we are using a static regression formula, with stationary series, whose moments do not change with time. This makes so that there is no space for differentiation of the betas along the time. This would also be in line with the low difference between coefficients, as seen before.

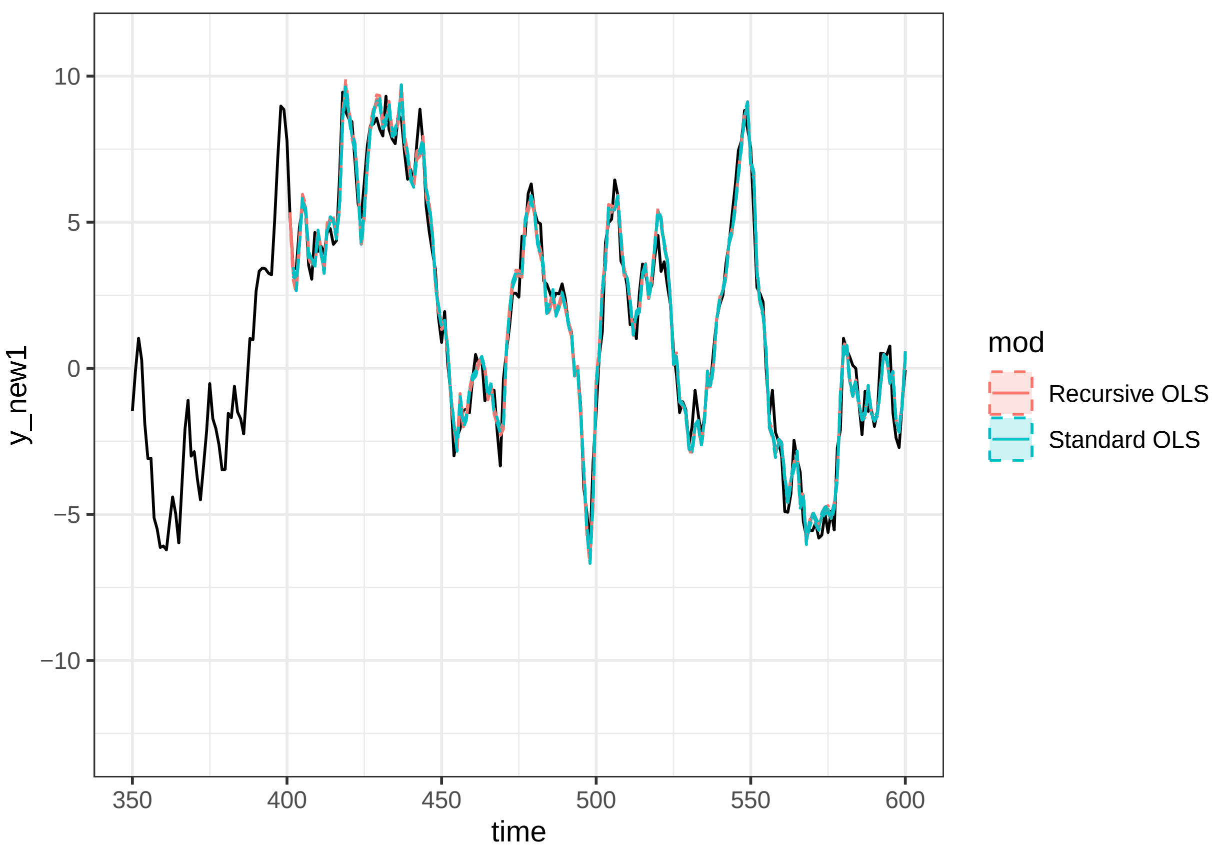
2.3 Q5. OLS With Dynamic

We can re-do the same procedure that was made in the above section, but using `y_new1 ~ x_new + lag(x_new) + lag(y_new1)` as the `lm()` formula.

```r
# Standard OLS
mod_std <- lm(y_new1 ~ x_new + lag(x_new) + lag(y_new1), data[101:n,])

# Predictions:
pred_std <- predict(mod_std, data[(n - 199):n,], se.fit = TRUE)[1:2] %>%
  bind_cols() %>%
  mutate(mod = "Standard OLS", time = (n - 199):n)

# Recursive OLS
mod_rec <- list()
pred_rec <- list()

for (i in 1:200) {
  mod_rec[[i]] <- lm(y_new1 ~ x_new + lag(x_new) + lag(y_new1), data[101:(n - 201 + i),])
  pred_rec[[i]] <- predict(mod_rec[[i]], data[(n - 201 + i):(n - 201 + i + 1),], se.fit = TRUE)[1:2] %>
    map(~.x[2])
}

# Predictions:
pred_rec <- pred_rec %>%
  bind_rows() %>%
  mutate(mod = "Recursive OLS", time = (n - 199):n)

# Coefficients:
tibble(
  beta = map_dbl(mod_rec, ~ .x$coefficients["x_new"]),
  time = (n - 199):n
) %>%
  ggplot(aes(time, beta)) +
  geom_line() +
  geom_hline(yintercept = mod_std$coefficients["x_new"], color = "red")
```



```
# Predictions:
pred <- bind_rows(pred_std, pred_rec)

ggplot(data, aes(time)) +
  geom_line(aes(y = y_new1)) +
  geom_line(aes(y = fit, color = mod), pred) +
  geom_ribbon(aes(ymin = fit - 1.96*se.fit, ymax = fit + 1.96*se.fit, color = mod, fill = mod),
    pred, linetype = 2, alpha = 0.2
  ) +
  xlim(350, 600)
```



```
map_dfr(c("Standard OLS", "Recursive OLS"), function(x) {
  error <- filter(data, time >= 401)$y_new1 - filter(pred, mod == x)$fit
  list(
    Model = x,
    RMSE = (sqrt(sum(error^2, na.rm = TRUE)) / 200) %>% round(5),
    MAE = (sum(abs(error), na.rm = TRUE) / 200) %>% round(5)
  )
}) %>%
  stargazer(summary = FALSE, header = FALSE, table.placement = "H")
```

Table 5:

	Model	RMSE	MAE
1	Standard OLS	0.04559	0.52451
2	Recursive OLS	0.04609	0.53292

Again we can see the comparison of coefficients, predictions, and prediction errors.

Again, the moments of the series being constant yielded similar results with the standard and recursive OLS. This is in line with the conclusion taken before.

3 Part 2

3.1 Q1. Residuals

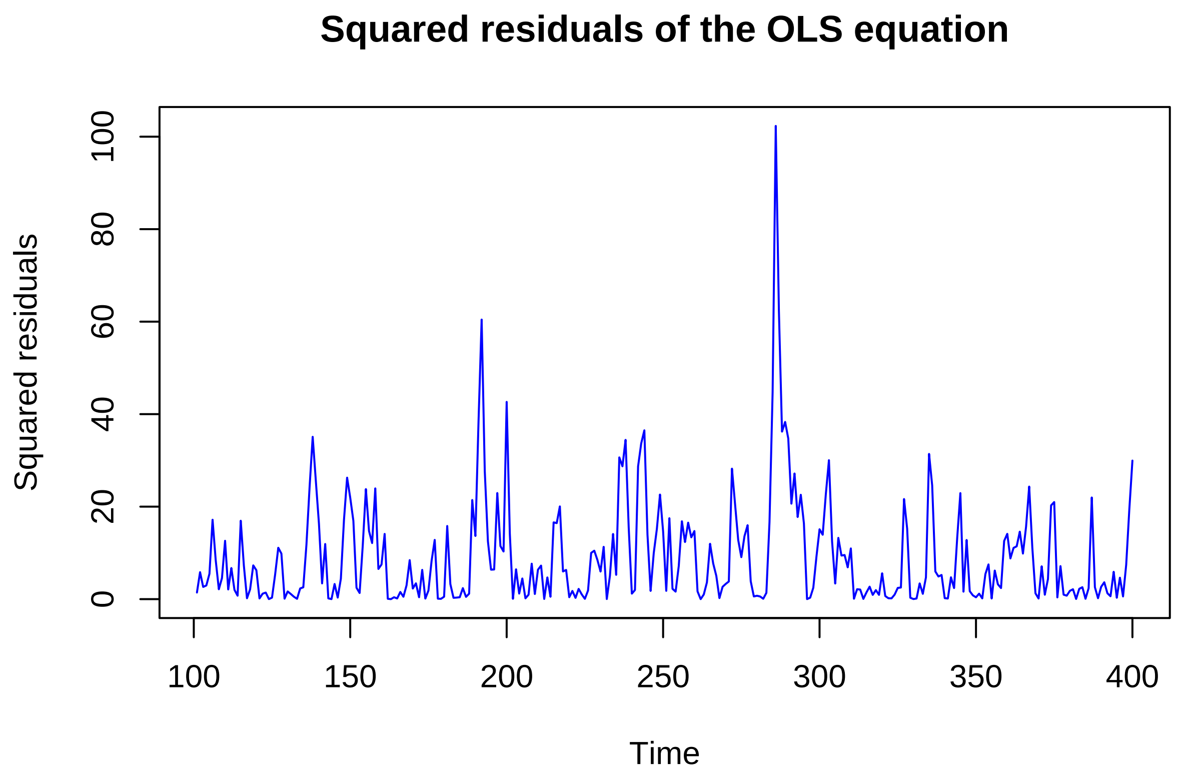
We estimated the following model, burning the first 100 observations and removing the last 200 observations:

$$y_t = \beta x_t + \epsilon_t$$

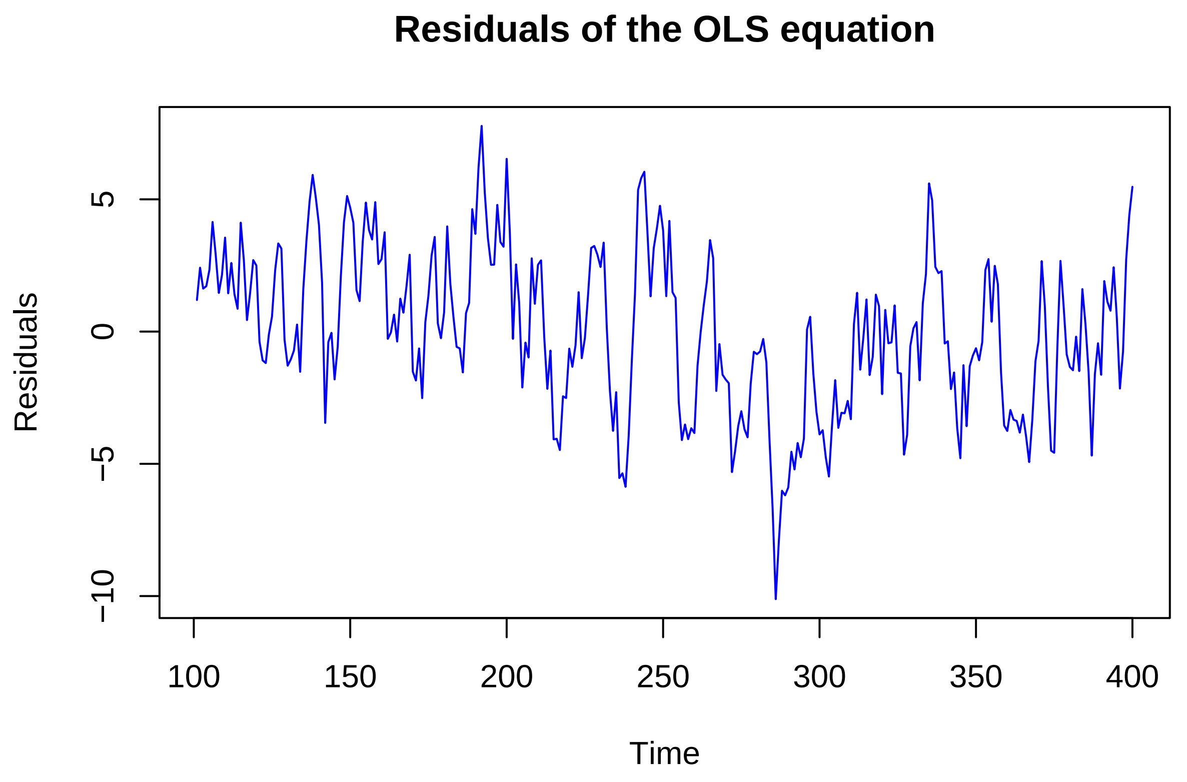
```
# Adjusting the dataset
dados <- read_csv("chap1_g5.csv")
dados_completos <- dados # keeping the complete dataset
dados <- tail(dados, -100) # removing the first 100 observations
dados <- head(dados, -200) # removing the last 200 observations
row.names(dados) <- seq(1, nrow(dados)) # renaming the rows

# estimating the model as described above
modelo <- lm(y_new1 ~ x_new, data = dados)
```

Below we plot the the squared residuals and the residuals of the estimated equation, eventhough the latter was not required by the exercise.

```
# Getting the residuals of the model
residuos <- resid(modelo)

# squaring the residuals
residuos_quadrado <- residuos^2

# Chart of the squared residuals
plot(dados$time, residuos_quadrado, type = "l", col = "blue",
      main = "Squared residuals of the OLS equation",
      xlab = "Time", ylab = "Squared residuals")
```



```r
# Chart of the residuals
plot(dados$time, residuos, type = "l", col = "blue",
      main = "Residuals of the OLS equation",
      xlab = "Time", ylab = "Residuals")
```



The graphs above depict the regression residuals and squared residuals, which are essential for diagnosing potential issues in the regression model.

A well-specified model should have residuals that are centered around zero and randomly distributed. If residuals show systematic deviations from zero, it may indicate that the model fails to capture all relevant relationships, possibly due to incorrect model specification or other statistical issue.

In the second graph, the residuals do not seem to be centered around zero, particularly before and after the observation 250. This suggests that the model may not adequately capture the underlying data structure.

The first graph, showing squared residuals, is crucial for detecting heteroscedasticity (where the variance of the residuals changes with time). While sometimes it can be challenging to spot heteroscedasticity visually, there appears to be a mild indication of this issue in the latter half of our sample. However, it's important to note that model misspecification can sometimes lead to the incorrect conclusions about heteroscedasticity.

Next, we will conduct additional tests to better identify and address any potential issues in the estimation performed using the initial OLS model. It's worth considering that the results obtained below may be influenced by potential specification errors in the model, which could affect the conclusions drawn.

3.2 Q2. Tests

3.2.1 Goldfeld-Quant test

The Goldfeld-Quandt test is used to detect heteroscedasticity in a regression model by comparing the variances of residuals between two subgroups of data. To perform the test, the data is first sorted according to an independent variable suspected to influence the variance of the residuals. The dataset is then split into two parts. Separate regressions are run for each subgroup, and the residual sum of squares (RSS) for each is calculated. An F-test is then conducted to compare the variances of these residuals.

Rejecting the null hypothesis in the F-test indicates differing variances between the two groups, suggesting the presence of heteroscedasticity.

Here is how the test was conducted and its results:

```r
# Performing the G-Q test: splitting the sample around the observation 250
# which is were we suspect that there was a change in the variance of the residuals
# Note: this package only allows the sample to be split in 2 tranches of the same
# size. If we suspected that the change in variance occured in another point, the
# test would have to be conducted manually
gq_test <- gqtest(modelo, order.by = ~ time, data = dados)
print(gq_test)
```

```
##
##  Goldfeld-Quandt test
##
## data:  modelo
## GQ = 1.1358, df1 = 148, df2 = 148, p-value = 0.2198
## alternative hypothesis: variance increases from segment 1 to 2
```

The test did not reject the null hypothesis due to a high p-value, indicating that there is no evidence of heteroscedasticity.

3.2.2 Breusch-Pagan Test

The Breusch-Pagan test is used to detect heteroscedasticity in a regression model. It works by first fitting the original regression model and calculating the residuals. These residuals are then squared and used as the dependent variable in a secondary auxiliary regression with the original independent variables. The purpose of this step is to determine if the variance of the residuals is related to the independent variables. If the regression shows a significant relationship, it suggests that the residual variance is not constant, indicating heteroscedasticity.

The test statistic is calculated as $T \times R^2$, where T is the sample size and R^2 comes from the auxiliary discussed above. This statistic follows a chi-square distribution with degrees of freedom equal to the number of independent variables in the model.

Rejecting the null hypothesis in the test confirms the presence of heteroscedasticity in the model.

```r
# Performing a BP test
bp_test <- bptest(modelo)
print(bp_test)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  modelo
## BP = 0.91753, df = 1, p-value = 0.3381
```

In our example the test did not reject the null hypothesis due to a high p-value, indicating that there is no evidence of heteroscedasticity.

3.2.3 White Test

The White test is another method used to detect heteroscedasticity. It is similar to the BP test, but admits non-linear forms for the heterosexuality. The test works by first fitting the original regression model and calculating the residuals. These residuals are then used in an auxiliary regression where the squared residuals serve as the dependent variable. In this auxiliary regression, the independent variables from the original model, along with their squares and interaction terms, are used as predictors. In our case:

$$\hat{\epsilon_i}^2 = \gamma_0 + \gamma_1 x_i + \gamma_2 x_i^2 + u_i$$

Where: $\hat{\epsilon_i}^2$ represents the squared residuals of the model; x_i denotes the values of the independent variable; γ_0, γ_1, and γ_2 are the coefficients to be estimated; u_i is the error term of the auxiliary model.

The test statistic for the White test is calculated as $T \times R^2$, where T is the sample size and R^2 comes from the auxiliary regression. As in the BP test rejecting the null hypothesis confirms the presence of heteroscedasticity in the model.

```r
# Performing a White test
white_test <- bptest(modelo, ~ poly(x_new, 2) + I(x_new^2), data = dados)
print(white_test)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  modelo
## BP = 2.9876, df = 2, p-value = 0.2245
```

The test did not reject the null hypothesis due to a high p-value, indicating that there is no evidence of heteroscedasticity.

As mentioned earlier, even if we had rejected the null hypothesis and detected heteroscedasticity in the tests conducted so far, we still couldn't be certain that this is the case, as our model might also suffer from misspecification issues.

3.2.4 Jarque-Bera Test

The Jarque-Bera test is used to assess whether the residuals of a regression model follow a normal distribution. It tests the null hypothesis that the residuals are normally distributed. The test is based on the skewness and kurtosis of the residuals.

The Jarque-Bera test statistic is calculated using the formula:

$$JB = \frac{T}{6}\left(S^2 + \frac{(K-3)^2}{4}\right)$$

where T is the sample size, S is the skewness of the residuals, and K is the kurtosis of the residuals. This test statistic follows a chi-square distribution with 2 degrees of freedom.

```r
# Performing a JB test
jb_test <- jarque.bera.test(residuals(modelo))
print(jb_test)
```

```
##
##  Jarque Bera Test
##
## data:  residuals(modelo)
## X-squared = 2.3438, df = 2, p-value = 0.3098
```

The test did not rejected the null hypothesis due to a high p-value, indicating that the residuals are normally distributed.

3.2.5 Durbin-Watson Test

The Durbin-Watson test is used to detect the presence of autocorrelation in the residuals. Autocorrelation can be problematic as it violates the assumption of independence of errors in linear regression models.

To perform the test, we compute the Durbin-Watson statistic using the residuals from our regression model. This statistic is calculated as:

$$DW = \frac{\sum_{t=2}^{T}(\hat{\epsilon}_t - \hat{\epsilon}_{t-1})^2}{\sum_{t=1}^{T}\hat{\epsilon}_t^2}$$

where $\hat{\epsilon}_t$ represents the residual at time t. The null hypothesis states that there is no autocorrelation in the residuals.

```r
# Performing a DW test
dw_test <- dwtest(modelo)
print(dw_test)
```

```
##
##  Durbin-Watson test
##
## data:  modelo
## DW = 0.38864, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

Since we observe a low p-value, we reject the null hypothesis. Consequently, this indicates the presence of autocorrelation in the residuals.

3.2.6 Chow Test

The Chow test assesses structural changes in a regression model at a specific point or between groups by comparing coefficients. To perform the test, you estimate the model with the full dataset, then split the data into segments, and estimate the model separately for each segment. The test statistic compares the sum of squared residuals from the pooled and segmented models. Rejecting the null hypothesis indicates a structural break, meaning coefficients differ between segments. One limitation of the Chow test is that it requires pre-specifying where potential breaks occur. In this case, our choice of observation 250 to divide the sample into two halves is based on prior observations and analyses.

```r
# Performing a Chow test
chow_test <- sctest(y_new1 ~ x_new, type = "Chow", data = dados, from = 0.5)
print(chow_test)
```

```
##
##  Chow test
##
## data:  y_new1 ~ x_new
## F = 46.989, p-value < 2.2e-16
```

Since the p-value was small, we rejected the null hypothesis. This indicates that there is a structural break in the regression model at the specified point.To better pinpoint where the structural break occurred, we will proceed with another test for further analysis.

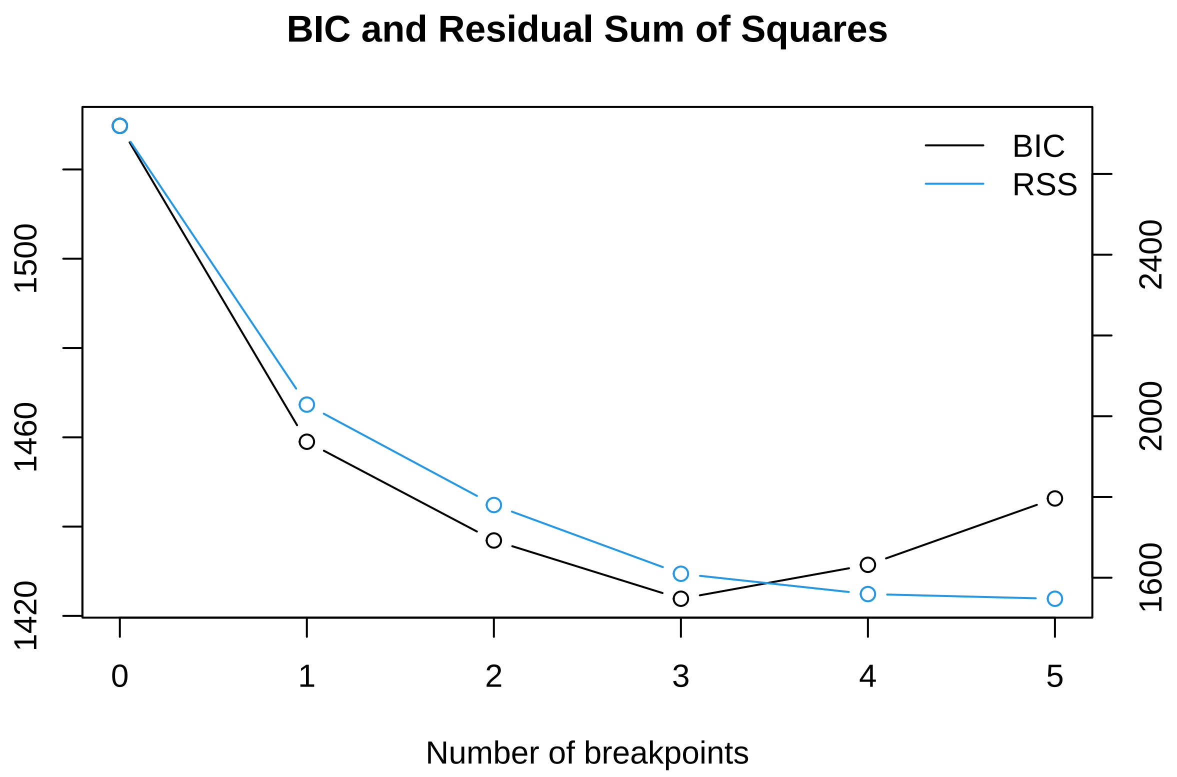
3.2.7 Bai-Perron Test

The Bai-Perron test is used to detect multiple structural breaks in a regression model at unknown points in time. Unlike the Chow test, which requires pre-specifying break points, the Bai-Perron test allows for the identification of break points within the data. To perform the test, you estimate the model allowing for multiple break points and compare the sum of squared residuals from the model with breaks to the sum of squared residuals from the model without breaks. The test statistic evaluates whether the introduction of breaks significantly improves the model fit. Rejecting the null hypothesis indicate the presence of structural changes at the identified break points, whereas non-significant results suggest no evidence of multiple structural breaks in the model.

```r
## Bai Perron Test
bp_test <- breakpoints(y_new1 ~ x_new, data = dados)
summary(bp_test)
```

```
##
##   Optimal (m+1)-segment partition:
##
## Call:
## breakpoints.formula(formula = y_new1 ~ x_new, data = dados)
##
## Breakpoints at observation number:
##
## m = 1           153
## m = 2           154 210
## m = 3       101 166 211
## m = 4    46 101 166 211
## m = 5    46 101 165 210 255
##
## Corresponding to breakdates:
##
## m = 1                                        0.51
## m = 2                                        0.513333333333333 0.7
## m = 3                     0.336666666666667 0.553333333333333 0.703333333333333
## m = 4    0.153333333333333 0.336666666666667 0.553333333333333 0.703333333333333
## m = 5    0.153333333333333 0.336666666666667 0.55              0.7
##
## m = 1
## m = 2
## m = 3
## m = 4
## m = 5    0.85
##
## Fit:
##
## m    0    1    2    3    4    5
## RSS 2719 2029 1780 1610 1560 1548
## BIC 1530 1459 1437 1424 1431 1446
```

```r
# Mostrar os pontos de quebra
plot(bp_test)
```



The test identified three structural breaks at observations 201, 266, and 311, as per the BIC. Please note that the number of these breakpoints differ from the output reported above, as the test was conducted on a reduced sample size as indicated by the exercise.

3.3 Q3. Dummy Variables

Below we estimate the model with dummies that represent the breakpoints found in the previous exercise:

$$y_t = \beta_0 + \beta_1 x_t + \gamma_1 D_{1t} + \gamma_2 D_{2t} + \gamma_3 D_{3t} + (\delta_1 x_t \cdot D_{1t} + \delta_2 x_t \cdot D_{2t} + \delta_3 x_t \cdot D_{3t})$$

We then estimate this model, and forecast for the last 200 periods of our sample:

```
## Estimating a model with dummies from the previous test
# 3 breakes were obtained, so we will have 3 dummies
# these dummies will also interact with x, so we will estimate 8 parameters

# Dummies trimmed
  dummy1 <- ifelse(dados_completos$time >= 201, 1, 0)
  dummy1_trim <- tail(dummy1, -100)
  dummy1_trim <- head(dummy1_trim, -200)

  dummy2 <- ifelse(dados_completos$time >= 266, 1, 0)
  dummy2_trim <- tail(dummy2, -100)
  dummy2_trim <- head(dummy2_trim, -200)

  dummy3 <- ifelse(dados_completos$time >= 311, 1, 0)
  dummy3_trim <- tail(dummy3, -100)
  dummy3_trim <- head(dummy3_trim, -200)

  dados <- cbind(dados, dummy1_trim)
  dados <- cbind(dados, dummy2_trim)
  dados <- cbind(dados, dummy3_trim)

# Model estimation OLS
model_with_breaks <- lm(y_new1 ~ x_new + dummy1_trim + dummy2_trim + dummy3_trim +
+ x_new*dummy1_trim + x_new*dummy2_trim + x_new*dummy3_trim, data = dados)

# Forecast
proj <- model_with_breaks$coefficients[1] +
  + model_with_breaks$coefficients[2]*dados_completos$x_new +
  + model_with_breaks$coefficients[3]*
  ifelse(dados_completos$time > 201, 1, 0) +
  + model_with_breaks$coefficients[4]*
  ifelse(dados_completos$time > 266, 1, 0) +
  + model_with_breaks$coefficients[5]*
  ifelse(dados_completos$time > 311, 1, 0) +
  + model_with_breaks$coefficients[6]*
  ifelse(dados_completos$time > 201, 1,0)*
  dados_completos$x_new +
  + model_with_breaks$coefficients[7]*
  ifelse(dados_completos$time > 266, 1, 0)*
  dados_completos$x_new +
  + model_with_breaks$coefficients[8]*
  ifelse(dados_completos$time > 311, 1, 0)*
  dados_completos$x_new


# Calculate RMSE and MAE
rmse_proj <- sqrt(mean((tail(proj, 200) - tail(dados_completos$y_new1, 200))^2))
mae_proj <-  mean(abs(tail(proj, 200) - tail(dados_completos$y_new1, 200)))

# Recursive forecast
proj_recursive <- dados_completos$y_new1

for (i in 0:199) {
  dados <- dados_completos
  dados <- tail(dados, -100)
  dados <- head(dados, -200+i)
  row.names(dados) <- seq(1, nrow(dados))

  dummy1 <- ifelse(dados_completos$time >= 201, 1, 0)
  dummy1_trim <- tail(dummy1, -100)
  dummy1_trim <- head(dummy1_trim, -200+i)

  dummy2 <- ifelse(dados_completos$time >= 266, 1, 0)
  dummy2_trim <- tail(dummy2, -100)
  dummy2_trim <- head(dummy2_trim, -200+i)

  dummy3 <- ifelse(dados_completos$time >= 311, 1, 0)
  dummy3_trim <- tail(dummy3, -100)
  dummy3_trim <- head(dummy3_trim, -200+i)

  dados <- cbind(dados, dummy1_trim)
  dados <- cbind(dados, dummy2_trim)
  dados <- cbind(dados, dummy3_trim)

  model_with_breaks <- lm(y_new1 ~ x_new + dummy1_trim +
                               dummy2_trim + dummy3_trim +
+ x_new*dummy1_trim + x_new*dummy2_trim + x_new*dummy3_trim,
data = dados)


  proj <- model_with_breaks$coefficients[1] +
    model_with_breaks$coefficients[2]*dados_completos$x_new +
    + model_with_breaks$coefficients[3]*
    ifelse(dados_completos$time > 201, 1, 0) +
    + model_with_breaks$coefficients[4]*
    ifelse(dados_completos$time > 266, 1, 0) +
    + model_with_breaks$coefficients[5]*
    ifelse(dados_completos$time > 311, 1, 0) +
    + model_with_breaks$coefficients[6]*
    ifelse(dados_completos$time > 201, 1, 0)*dados_completos$x_new +
    + model_with_breaks$coefficients[7]*
    ifelse(dados_completos$time > 266, 1, 0)*dados_completos$x_new +
    + model_with_breaks$coefficients[8]*
    ifelse(dados_completos$time > 311, 1, 0)*dados_completos$x_new

  proj_recursive[401+i] <- proj[401+i]
  }


# Calculate RMSE and MAE
rmse_proj_recursive <- sqrt(mean((tail(proj_recursive, 200) - tail(dados_completos$y_new1, 200))^2))
mae_proj_recursive <-  mean(abs(tail(proj_recursive, 200) - tail(dados_completos$y_new1, 200)))

# Print results
summary(model_with_breaks)
```

```
##
## Call:
## lm(formula = y_new1 ~ x_new + dummy1_trim + dummy2_trim + dummy3_trim +
##     +x_new * dummy1_trim + x_new * dummy2_trim + x_new * dummy3_trim,
##     data = dados)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8794 -1.7009  0.1032  1.8179  7.0455
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.69595    0.30993   8.699  < 2e-16 ***
## x_new                1.44548    0.19254   7.508 2.86e-13 ***
## dummy1_trim         -2.38012    0.43934  -5.418 9.48e-08 ***
## dummy2_trim       -4.35602    0.61427  -7.091 4.66e-12 ***
## dummy3_trim        4.64445    0.55006   8.444 3.49e-16 ***
## x_new:dummy1_trim  0.02778    0.25481   0.109   0.9132
## x_new:dummy2_trim -0.07030    0.31671  -0.222   0.8244
## x_new:dummy3_trim  0.60247    0.28598   2.107   0.0357 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.51 on 491 degrees of freedom
## Multiple R-squared:  0.6902, Adjusted R-squared:  0.6858
## F-statistic: 156.3 on 7 and 491 DF,  p-value: < 2.2e-16
```

```
print(rmse_proj)
```

```
## [1] 3.05062
```

```
print(rmse_proj_recursive)
```

```
## [1] 2.741597
```

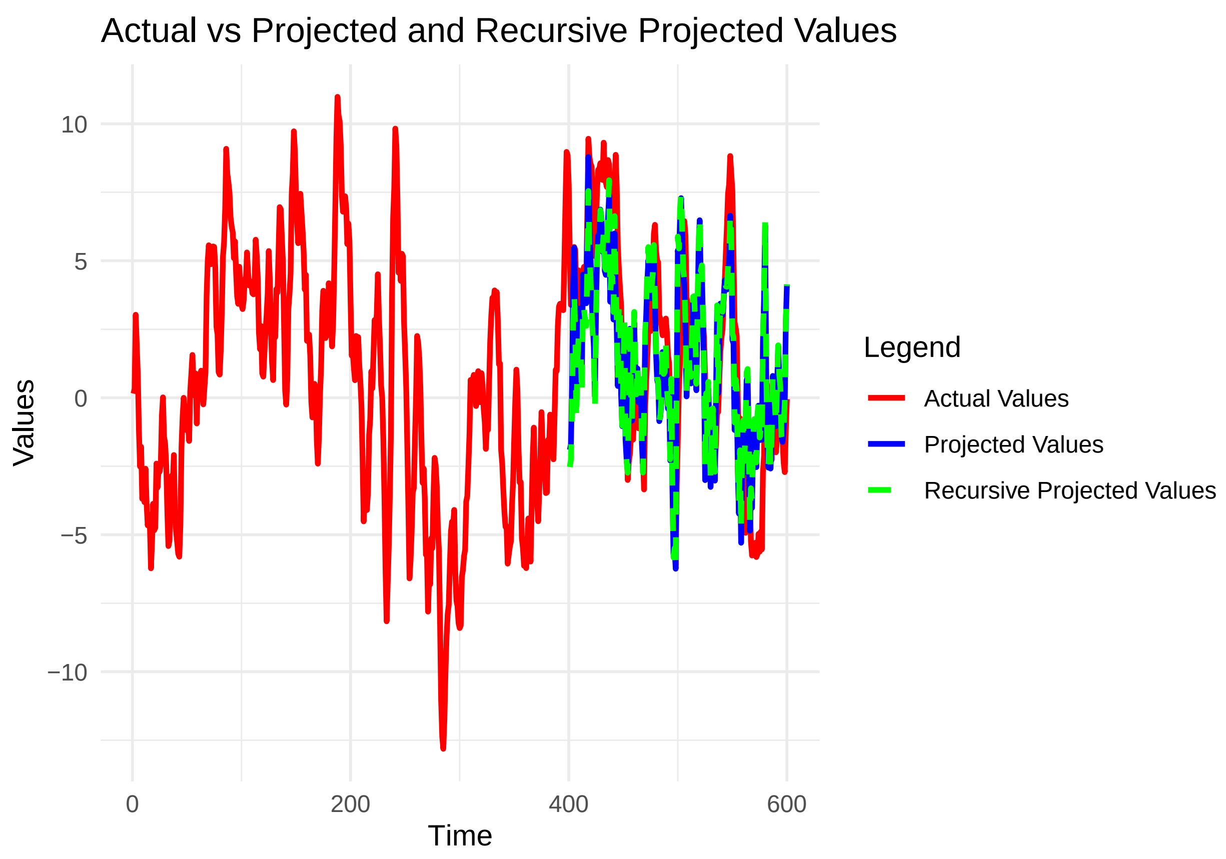
```
print(mae_proj)
```

```
## [1] 2.534061
```

```
print(mae_proj_recursive)
```

```
## [1] 2.262207
```

```
# Create a data frame for plotting
plot_data <- data.frame(
  Time = dados_completos$time,
  Proj = proj,
  Actual = dados_completos$y_new1,
  Proj_Recursive = proj_recursive  # Add your recursive projections here
)

# Subset the data for Time >= 401 for projected and recursive projected values
proj_subset <- plot_data[plot_data$Time >= 401, ]

# Plot actual values, projected values, and recursive projected values
ggplot() +
  geom_line(data = plot_data, aes(x = Time, y = Actual, color = "Actual Values"), size = 1) +
  geom_line(data = proj_subset, aes(x = Time, y = Proj,
                                   color = "Projected Values"), size = 1) +
  geom_line(data = proj_subset, aes(x = Time, y = Proj_Recursive,
  color = "Recursive Projected Values"), size = 1, linetype = "dashed") +
  labs(title = "Actual vs Projected and Recursive Projected Values",
       x = "Time",
       y = "Values") +
  scale_color_manual(name = "Legend",
  values = c("Actual Values" = "red", "Projected Values" = "blue",
             "Recursive Projected Values" = "green")) +
  theme_minimal()
```



As expected, the errors (both MAE and RMSE) are smaller for the recursive projections, as they more accurately capture the dynamics at the margin by continuously re-estimating the parameters.