Forecasting Task - Problem Set 2

Bernardo Calvente e Ricardo Castro

2024-08-25

1 Setup

The following packages were used:

```
library(tidyverse)
library(glue)
library(broom)

library(nlme)
library(lmtest)
library(strucchange)

library(knitr)
library(kableExtra)

theme_set(theme_bw())
```

We've created a custom function to get the information criteria of a model, divided by its degrees of freedom:

```
get_info <- function(model, n = 2) {
  df.residual.gls <- \(object, ...) object$dims$N
  df <- df.residual(model)

  info <- round(c(AIC(model), BIC(model)) / df, n)
  tibble(`AIC/DF` = info[1], `BIC/DF` = info[2])
}
```

Custom function to get test results:

```
tidy.breakpointsfull <- function(x, ...) {
  tibble(
    statistic = sum(!is.na(x$breakpoints)),
    p.value = NA
  )
}

test_to_table <- function(test, mods = results) {
  tryCatch({
    with(tidy(test), glue("{round(statistic, 2)} ({round(p.value, 2)})"))
  },
    error = function(e) NA
  )
}
```

The exercise did not explicitly specify which dataset to use, so we are proceeding with the same dataset that

we used in Problem Set 1. We also reduced estimation sample as in Pset1.

```
data <- read_csv("../ps1/ps1_data.csv") %>%
  transmute(
    time = time,
    x0 = x_new,
    y0 = y_new1,
    x1 = lag(x_new, n = 1),
    y1 = lag(y_new1, n = 1),
    d1x0 = x0 - x1,
    d1y0 = y0 - y1
  )

data_est <- slice(data, 101:(nrow(data) - 200))
n <- nrow(data_est)
```

2 Models

2.1 Setup

Before we define the functional form of the models, we need to define the residuals from the "static" model, will be used on the ECM:

```
mod_static <- lm(y0 ~ x0, data_est)
data_est$StaticResid <- c(NA, mod_static$residuals[-n])
```

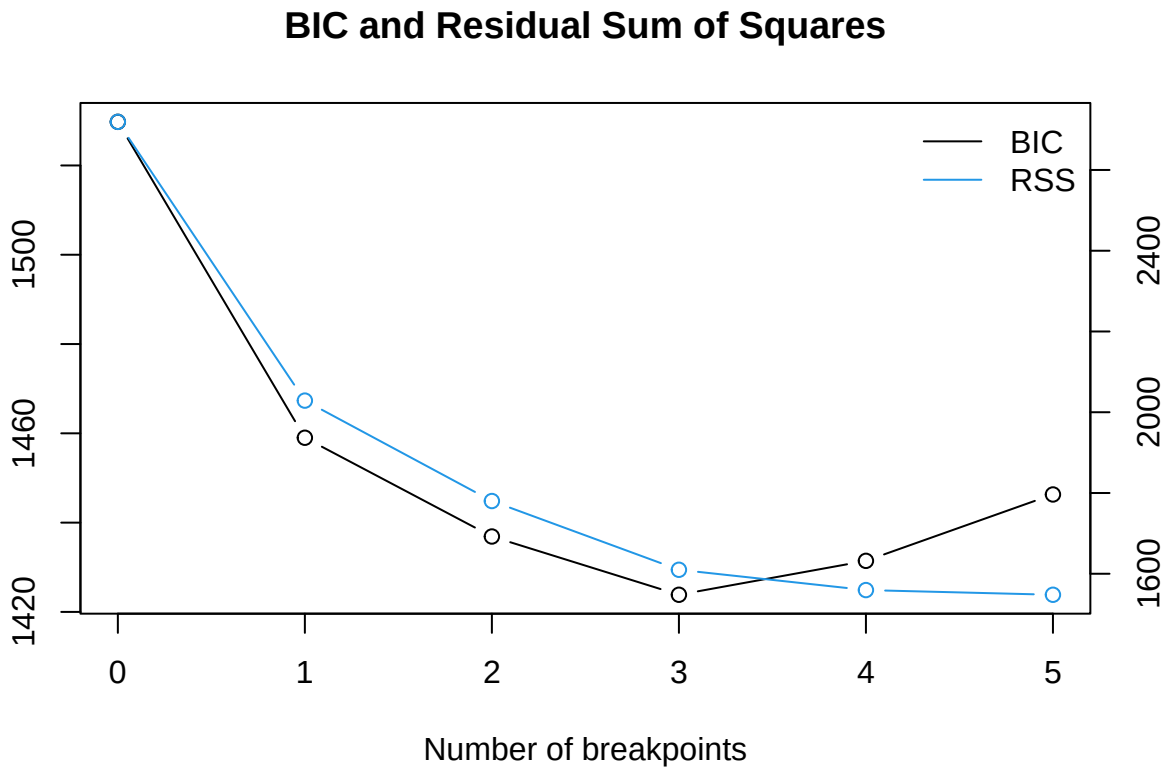
To ensure our models effectively capture potential structural changes, we will To ensure our models effectively capture potential structural changes, we will begin by defining the necessary dummy variables, following the same approach as in the first problem set. We will utilize the Bai and Perron test to identify structural breakpoints within the data. This test is designed to detect multiple breakpoints by analyzing shifts in the underlying data relationships, allowing us to precisely pinpoint where these changes occur and adjust our models accordingly with the appropriate dummy variables.

Same procedure as in the 1st problem set (Bai and Perron test). We chart the BIC and AIC and number of break points.

```
bp_test <- breakpoints(y0 ~ x0, data = data_est)
print(bp_test)
```

```
##
##   Optimal 4-segment partition:
##
## Call:
## breakpoints.formula(formula = y0 ~ x0, data = data_est)
##
## Breakpoints at observation number:
## 101 166 211
##
## Corresponding to breakdates:
## 0.3366667 0.5533333 0.7033333
```

```
plot(bp_test)
```



Similar to the previous problem set, we have identified potential breakpoints at observations 201, 266, and 311. Below, we create dummy variables corresponding to these specific points:

```
data_est <- data_est %>%
  mutate(
    D201 = ifelse(row_number() >= 101, 1, 0),
    D266 = ifelse(row_number() >= 166, 1, 0),
    D311 = ifelse(row_number() >= 211, 1, 0)
  )
```

2.2 Models Definition

Now, we can proceed to define the models, outlining the modelling approach, their respective formulas, and any additional arguments required. The models under consideration include:

- Unrestricted $ADL(1,1)$ (as discussed in Lecture 3):

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 x_t + \beta_3 x_{t-1} + \epsilon_t$$

- Static regression model:

$$y_t = \beta_0 + \beta_1 x_t + \epsilon_t$$

- $AR(1)$ model:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t$$

- Model with differences (i.e., "rate of change' '):

$$\Delta y_t = \beta_0 + \beta_1 \Delta x_t + \epsilon_t$$

- Leading indicator model (with x_{t-1}, as in class):

$$y_t = \beta_0 + \beta_1 x_{t-1} + \epsilon_t$$

- Distributed lags model:

$$y_t = \beta_0 + \beta_1 x_t + \beta_2 x_{t-1} + \epsilon_t$$

- Partial adjustment model:

$$y_t = \beta_0 + \beta_1 x_t + \beta_2 y_{t-1} + \epsilon_t$$

- Dead start model (i.e., "reduced form' '):

$$y_t = \beta_0 + \beta_1 x_{t-1} + \beta_2 y_{t-1} + \epsilon_t$$

- Static regression with $AR(1)$ errors:

$$y_t = \beta_0 + \beta_1 y_{t-1} + u_t$$

$$u_t = \rho u_{t-1} + \epsilon_t$$

- $ADL(1,1)$ with the dummy variables:

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 x_t + \beta_3 x_{t-1} + \beta_4 D_{1t} + \beta_5 D_{2t} + \beta_6 D_{3t} + \beta_7 D_{1t} x_t + \beta_8 D_{2t} x_t + \beta_9 D_{3t} x_t \epsilon_t$$

Obs: for simplicity, we have included dummies only at the level and x_t.

- Equilibrium Correction Model:

$$\Delta y_t = \beta_0 + \beta_1 \Delta x_t + (\beta_3 - 1)(x_{t-1} - \beta_4 - \beta_5 x_{t-1}) + \epsilon_t$$

Obs: $ADL(1,1)$, in case variables are $I(1)$, needs to be reparametrized into the EQCM.

```
models <- list(
  ADL11 = list(lm, y0 ~ y1 + x0 + x1),
  Static = list(lm, y0 ~ x0),
  AR1 = list(lm, y0 ~ y1),
  Diff = list(lm, y0 ~ offset(1 * y1) + d1x0),
  LeadInd = list(lm, y0 ~ y1 + x1),
  DistLags = list(lm, y0 ~ x0 + x1),
  PartAdj = list(lm, y0 ~ y1 + x0),
  DeadStart = list(lm, y0 ~ y1 + x1),
  StaticAR1 = list(gls, y0 ~ x0, correlation = corARMA(p = 1, q = 0)),
  ADL11D = list(lm, y0 ~ y1 + x0 + x1 + D201*x0 + D266*x0 + D311*x0),
  ECQM = list(lm, y0 ~ offset(1 * y1) + d1x0 + StaticResid)
)
```

Now, we can estimate each one, with a functional approach.

```
results <- map(models, function(l) {
  l$data <- data_est[2:n,]
  do.call(l[[1]], l[-1])
})
```

2.3 Results

With the models now estimated, we can apply our custom function to compute the information criteria. The summarized results are presented in Tables 1 and 2.

```
results_table <- imap(list(1:6, 7:11), function(ind, name) {
  modelsummary::modelsummary(results[ind],
    output = "kableExtra",
    gof_omit = "(^Log\\.Lik\\.$)|(^F$)|(^AIC$)|(^BIC$)",
    gof_function = get_info,
    starts = TRUE,
    title = "Model results",
  )
})

kable_styling(results_table[[1]], latex_options = c("HOLD_position"))
```

Table 1: Model results

	ADL11	Static	AR1	Diff	LeadInd	DistLags
(Intercept)	0.043	0.207	0.024	0.015	0.050	0.197
	(0.035)	(0.175)	(0.074)	(0.061)	(0.061)	(0.143)
y1	0.796		0.966		0.786	
	(0.012)		(0.016)		(0.020)	
x0	0.838	2.096				0.763
	(0.033)	(0.103)				(0.138)
x1	−0.043				0.642	1.681
	(0.042)				(0.056)	(0.138)
d1x0				0.672		
				(0.055)		
Num.Obs.	299	299	299	299	299	299
R2	0.984	0.585	0.927	0.332	0.949	0.724
R2 Adj.	0.984	0.583	0.926	0.330	0.949	0.722
RMSE	0.60	3.01	1.27	1.04	1.05	2.46
AIC/DF	1.86	5.1	3.36	2.96	3	4.71
BIC/DF	1.92	5.14	3.39	3	3.05	4.76

```
kable_styling(results_table[[2]], latex_options = c("HOLD_position"))
```

Table 2: Model results

	PartAdj	DeadStart	StaticAR1	ADL11D	ECQM
(Intercept)	0.044	0.050	1.205	0.125	0.011
	(0.035)	(0.061)	(1.993)	(0.085)	(0.049)
y1	0.789	0.786		0.782	
	(0.009)	(0.020)		(0.015)	
x0	0.815		0.675	0.841	
	(0.025)		(0.056)	(0.051)	
x1		0.642		−0.029	
		(0.056)		(0.043)	
D201				−0.114	
				(0.111)	
D266				−0.079	
				(0.162)	
D311				0.126	
				(0.152)	
x0 × D201				0.012	
				(0.061)	
x0 × D266				0.042	
				(0.075)	
x0 × D311				−0.142	
				(0.076)	
d1x0					0.655
					(0.045)
StaticResid					−0.207
					(0.016)
Num.Obs.	299	299	299	299	299
R2	0.984	0.949	0.274	0.984	0.568
R2 Adj.	0.984	0.949		0.984	0.565
RMSE	0.60	1.05	3.98	0.59	0.84
AIC/DF	1.85	3	2.95	1.9	2.54
BIC/DF	1.9	3.05	3	2.05	2.59

Based on both the BIC and AIC criteria, the Partial Adjustment model demonstrated the best performance. These criteria evaluate models by balancing goodness of fit with model complexity, penalizing the inclusion of unnecessary parameters. The Partial Adjustment model was likely favored because it effectively captured the key dynamics of the data while maintaining a parsimonious structure, making it the optimal choice among the models considered.

2.4 Tests

Next, we evaluate the validity of some assumptions underlying the models. We will loop through each model and apply a series of diagnostic tests: Breusch-Pagan, White, Jarque-Bera, Durbin-Watson, Chow, and Bai-Perron. The results are summarized in a table, displaying only p-values (for economy) for all tests except the Bai-Perron, where we instead report the number of breakpoints determined by the BIC. The null hypotheses for each test are as follows:

- **Breusch-Pagan**: The residuals have constant variance (homoscedasticity).
- **White**: The residuals are homoscedastic and independent of the regressors.
- **Jarque-Bera**: The residuals follow a normal distribution.
- **Durbin-Watson**: There is no autocorrelation in the residuals.

- **Chow**: There is no structural break at the specified point ($t = 250$) in the data.

```
imap_dfr(results, function(res, name) {
  poly <- glue("I({names(res$coefficients[-1])}^2)") %>%
    `[`(!grepl(":", .)) %>%
    paste(collapse = " + ") %>%
    paste(". ~ . +", .) %>%
    as.formula()

  tests <- list(
    BP = bptest(res$terms, data = data_est[2:n,]),
    White = bptest(update.formula(res$terms, poly), data = data_est[2:n,]),
    JB = tseries::jarque.bera.test(residuals(res)),
    DW = try(dwtest(res)),
    Chow = sctest(res$terms, type = "Chow", data = data_est, from = 0.5),
    BaiPerron = breakpoints(res$terms, data = data_est)
  )

  map_dfc(tests, test_to_table) %>%
    mutate(Model = name, .before = 1)
}) %>%
  kable()
```

```
## Error in as.data.frame.default(data) :
##   cannot coerce class 'c("glsStruct", "modelStruct")' to a data.frame
```

Model	BP	White	JB	DW	Chow	BaiPerron
ADL11	3.89 (0.27)	6.16 (0.41)	3.12 (0.21)	2.13 (0.85)	1.11 (0.35)	0 (NA)
Static	0.88 (0.35)	4.13 (0.13)	2.34 (0.31)	0.39 (0)	46.99 (0)	3 (NA)
AR1	0.06 (0.81)	0.74 (0.69)	0.54 (0.76)	1.13 (0)	1.28 (0.28)	0 (NA)
Diff	0.07 (0.8)	1.15 (0.56)	0.37 (0.83)	0.11 (0)	78.36 (0)	3 (NA)
LeadInd	1.37 (0.5)	7.49 (0.11)	0.89 (0.64)	1.95 (0.29)	4.6 (0)	0 (NA)
DistLags	1.49 (0.48)	6.88 (0.14)	2.62 (0.27)	0.43 (0)	32.48 (0)	3 (NA)
PartAdj	3.01 (0.22)	5.13 (0.27)	3.14 (0.21)	2.12 (0.83)	1.15 (0.33)	0 (NA)
DeadStart	1.37 (0.5)	7.49 (0.11)	0.89 (0.64)	1.95 (0.29)	4.6 (0)	0 (NA)
StaticAR1	0.88 (0.35)	4.13 (0.13)	3.53 (0.17)	NA	46.99 (0)	3 (NA)
ADL11D	13.48 (0.14)	14.21 (0.29)	2.44 (0.29)	2.16 (0.83)	0.32 (0.97)	0 (NA)
ECQM	1.69 (0.43)	5.17 (0.27)	0.7 (0.7)	0.25 (0)	28.93 (0)	3 (NA)

From the table above, we can observe the following:

- **Heteroskedasticity - Breusch-Pagan and White tests**: None of the models rejected the null hypothesis, indicating that no significant evidence of heteroskedasticity was detected. However, it's important to consider that these tests might have limited power in the presence of model misspecification. If the models are incorrectly specified—such as omitting relevant variables or including irrelevant ones—this can lead to misleading results. The only exceptions were the models that involved differencing the variables, which may have introduced noise into the test, potentially affecting the results. Consequently, while the overall results suggest homoskedasticity, caution is warranted, and further diagnostic checks may be needed to ensure the robustness of the conclusions.
- **Normality - Jarque-Bera Test**: The test revealed non-normality exclusively in the differenced models, indicating that the differencing process may have introduced or amplified issues like skewness or kurtosis in the data. This is significant because differencing, while often used to address non-stationarity, can sometimes lead to distortions in the distribution of residuals, particularly if the original series was already stationary.

- **Autocorrelation - Durbin-Watson Test**: The test revealed autocorrelation in the residuals of some models, leading to the rejection of the null hypothesis. This outcome is significant because autocorrelation can compromise the validity of OLS regression results, potentially leading to biased and inefficient estimates. However, it is crucial to recognize that model misspecification, such as omitting important variables or incorrect model structure, can influence the Durbin-Watson test, causing it to detect autocorrelation that is actually due to these specification errors rather than genuine serial correlation.
- **Structural Breaks - Chow Test and Bai-Perron Test**: The test results were generally consistent, except for the dead start model and lead indicator model. In the distributed lags, difference, ECM and static models, the Bai-Perron test identified breakpoints.

2.4.1 Additional Tests

Here we test whether the series are stationary. This step is important, especially when using models that involve differencing the series, such as the "rate of change" model and the "equilibrium correction" model. Below, we perform two tests:

- **Augmented Dickey-Fuller (ADF) Test**: tests for the presence of a unit root in the series, which would indicate non-stationarity. The null hypothesis is the series has a unit root (i.e., it is non-stationary).
- **Phillips-Perron (PP) Test**: the test improves upon ADF by making non-parametric corrections for serial correlation and heteroskedasticity in the residuals, thus providing a more robust analysis in the presence of such issues. The null hypothesis is the series has a unit root (i.e., it is non-stationary).
- **F-test on** $ADL(1,1)$: we will also test whether the inclusion of the dummies from Problem Set 1 is necessary in our $ADL(1,1)$ model, resulting in the $ADL(1,1)$ + dummies model. Note that for simplicity we have only included dummies at the constant and x_t.
- **F-test on EQCM**: we test the joint hypothesis of the constant y in the long run (y_1's coefficient being equal to one), and the combined effect of x is zero.

```
# Function to perform unit root tests and return p-values
perform_tests <- function(series) {
  adf_pvalue <- aTSA::adf.test(series, 1, FALSE)$type1[,"p.value"]
  pp_pvalue <- tseries::pp.test(series)$p.value
  return(c(adf_pvalue, pp_pvalue))
}

# Apply the tests to each series in the data frame
data.frame(
  Test = c("ADF", "PP"),
  y = perform_tests(data_est$y0),
  x = perform_tests(data_est$x0)
) %>%
  mutate(across(-Test, ~round(.x, 3))) %>%
  kable(format = "latex", caption = "Unit Root Test P-Values") %>%
  kable_styling(latex_options = c("HOLD_position"))
```

Table 4: Unit Root Test P-Values

	Test	y	x
p.value	ADF	0.034	0.01
	PP	0.019	0.01

For x_t, the low p-value allows us to reject the null hypothesis of non-stationarity, indicating that the series is stationary. For y_t, while the ADF test suggested potential non-stationarity, the PP test indicated that the series is stationary. Given that the PP test is more robust to issues like serial correlation and heteroskedasticity,

we can reasonably conclude that y_t is likely stationary. Therefore, differencing the series may not be necessary for modeling.

Now, we run the last two tests.

```
nr <- length(resid(results$ADL11D))
fstat <- (sum(resid(results$ADL11)^2) / sum(resid(results$ADL11D)^2)) * ((nr - 7)/(nr - 4))

glue("P-value of ommited variable: {round(1 - pf(fstat, nr - 4, nr - 7))}")
```

```
## P-value of ommited variable: 0
```

Since the p-value from the F-test was not low enough to reject the null hypothesis, we conclude that the dummies do not significantly improve the model. This suggests that the simpler $ADL(1,1)$ model is sufficient, and adding breakpoints to the model do not provide any meaningful enhancement to the explanatory power. This result is in line with the observed on Chow and Bai-Perron tests.

```
car::linearHypothesis(results$ADL11, c("y1 = 1","x0 - x1 = 0"), test = "F") %>%
  tidy() %>%
  select(1:6) %>%
  knitr::kable()
```

term	null.value	estimate	std.error	statistic	p.value
y1	1	-0.2042395	0.0115709	381.0811	0
x0 - x1	0	0.8810010	0.0684563	381.0811	0

Lastly, we can see that both p-values are small, and the values for the coefficients are statistically different from what the null hypothesis stated.

3 Prediction

Below, we perform both static (i.e., fixed estimation sample) and recursive forecasts using the best model identified in the previous section (i.e., Partial adjustment model). We then plot the forecasts and calculate the associated forecast errors (i.e., RMSE and MAE).

```
# Model estimation OLS - BIC => best model was Partial adjustment per
model_best <- lm(y0 ~ y1 + x0, data = data_est)

# Forecast
proj_static <- model_best$coefficients[1] +
  model_best$coefficients[2] * data_est$y1 +
  model_best$coefficients[3] * data_est$x0


# Calculate RMSE and MAE for the last 200 obs (forecast sample)
rmse_proj <- sqrt(mean((tail(proj_static, 200) - tail(data_est$y0, 200))^2))
mae_proj <- mean(abs(tail(proj_static, 200) - tail(data_est$y0, 200)))

# Recursive forecast
proj_recursive <- data_est$y0

for (i in 0:199) {
  # Adjusting the data
  data_rec <- data %>% tail(-100) %>% head(-200 + i)
```

```
  # Estimating the model
  model_best <- lm(y0 ~ y1 + x0, data = data_rec)

  # Computing forecasts
  proj <- model_best$coefficients[1] +
    model_best$coefficients[2] * data_rec$y1 +
    model_best$coefficients[3] * data_rec$x0

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

# Calculate RMSE and MAE for recursive forecasts
rmse_proj_recursive <- sqrt(mean((tail(proj_recursive, 200) - tail(data_est$y0, 200))^2))
mae_proj_recursive <- mean(abs(tail(proj_recursive, 200) - tail(data_est$y0, 200)))

# Create a data frame for the comparison table
results_table <- data.frame(
  Metric = c("RMSE", "MAE"),
  Static = c(rmse_proj, mae_proj),
  Recursive = c(rmse_proj_recursive, mae_proj_recursive)
)

# Print results in a table format
kable(results_table, format = "latex", booktabs = TRUE, caption = "MAE and RMSE") %>%
  kable_styling(latex_options = c("HOLD_position"))
```

Table 6: MAE and RMSE

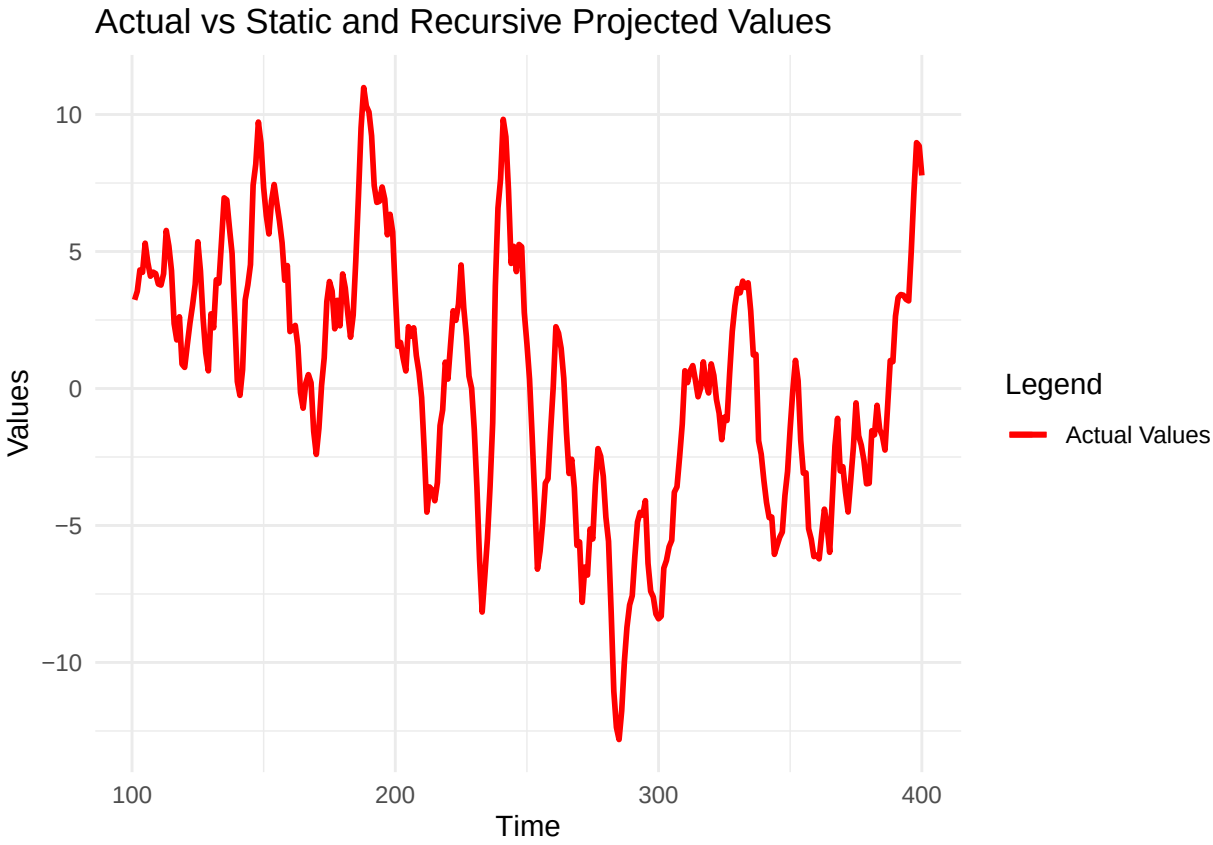
Metric	Static	Recursive
RMSE	0.5935022	NA
MAE	0.4809710	NA

```
# Create a data frame for plotting
plot_data <- data.frame(
  Time = data_est$time,
  Proj_Static = proj_static,
  Actual = data_est$y0,
  Proj_Recursive = proj_recursive
)

# 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_Static,
                                    color = "Static 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 Static and Recursive Projected Values",
       x = "Time",
```

```
      y = "Values") +
scale_color_manual(name = "Legend",
                   values = c("Actual Values" = "red",
                              "Static Projected Values" = "blue",
                              "Recursive Projected Values" = "green")) +
theme_minimal()
```



Based on the graphs and forecast errors, it is evident that the projections from both methodologies were quite close to each other. Interestingly, the static projection (using a constant estimation sample) had a lower forecast error, which is not typically expected, but can certainly occur.