Trabalho - Econometria IV

Guilherme Luz, Guilherme Masuko, Caio Garzeri

July 2023

```
library(lubridate)
library(tictoc)
library(zoo)
library(dynlm)  # for time series regressions
library(forecast)
```

Question 2

First of all, we must do some data wrangling.

```
# Import the data
raw_data = read_csv("data/2021-12.csv")
data0 = raw_data[-1, ] %>%
    select_if(~!any(is.na(.)))
transformation = raw_data[1, ]
```

The suggested transformations (in order to make the series stationary) are indicated according to the following numeration.

Transformation codes (from FRED):

1. no transformation
2. Δx_t
3. $\Delta^2 x_t$
4. $\log(x_t)$
5. $\Delta \log(x_t)$
6. $\Delta^2 \log(x_t)$
7. $\Delta(x_t/x_{t-1} - 1)$

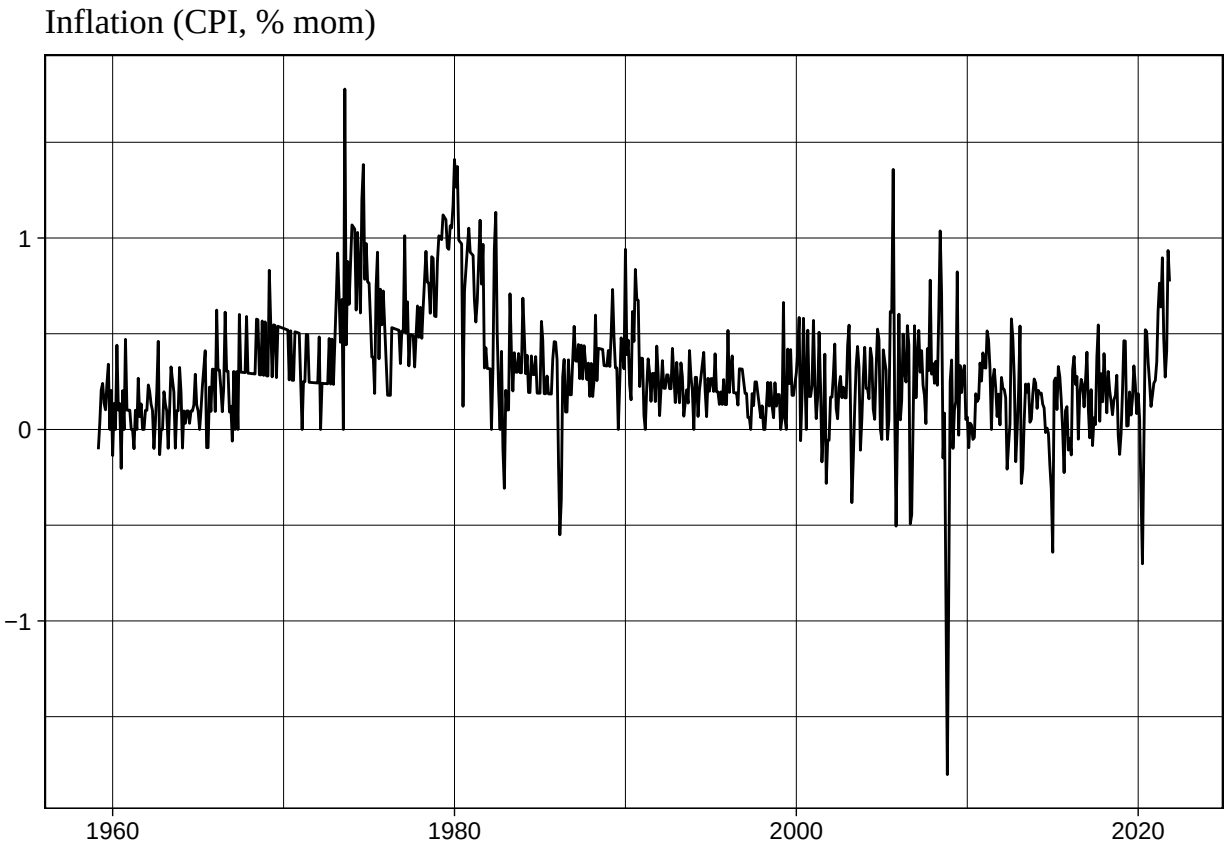
For the CPI, we apply a specific transformation to turn it into an inflation series. Detalhe: estou usando a definição de inflação dada no enunciado $\pi_t = \frac{\Delta P_t}{P_t}$, que difere da definição usual $\pi_t = \frac{\Delta P_t}{P_{t-1}}$

```
# Data transformations based on the FRED transformation
# codes
data = data0 %>%
    select(-sasdate) %>%
    rename(SP500 = "S&P 500", SPINDUST = "S&P: indust") %>%
    BVAR::fred_transform(type = "fred_md") %>%
    bind_cols(tibble(date = data0$sasdate[3:length(data0$sasdate)])) %>%
    mutate(date = as.Date(date, format = "%m/%d/%Y"))

# For the CPI, we transform into an inflation series
data = mutate(data, CPIAUCSL = 100 * (diff(data0$CPIAUCSL, differences = 1)/data0$CPIAUCSL[-1])[-1])
```

```
# Inflation as time series
inflation = data$CPIAUCSL %>%
    ts(start = c(year(data$date[1]), month(data$date[1])), frequency = 12)
```

The resulting inflation series, which we want to forecast is shown below.

```
# plot inflation

data %>%
    select(date, CPIAUCSL) %>%
    mutate(date = as.Date(date, format = "%m/%d/%Y")) %>%
    # filter(date>as.Date('2000-01-01')) %>%
ggplot(aes(x = date, y = CPIAUCSL)) + geom_line() + labs(title = "Inflation (CPI, % mom)",
    x = NULL, y = NULL)
```

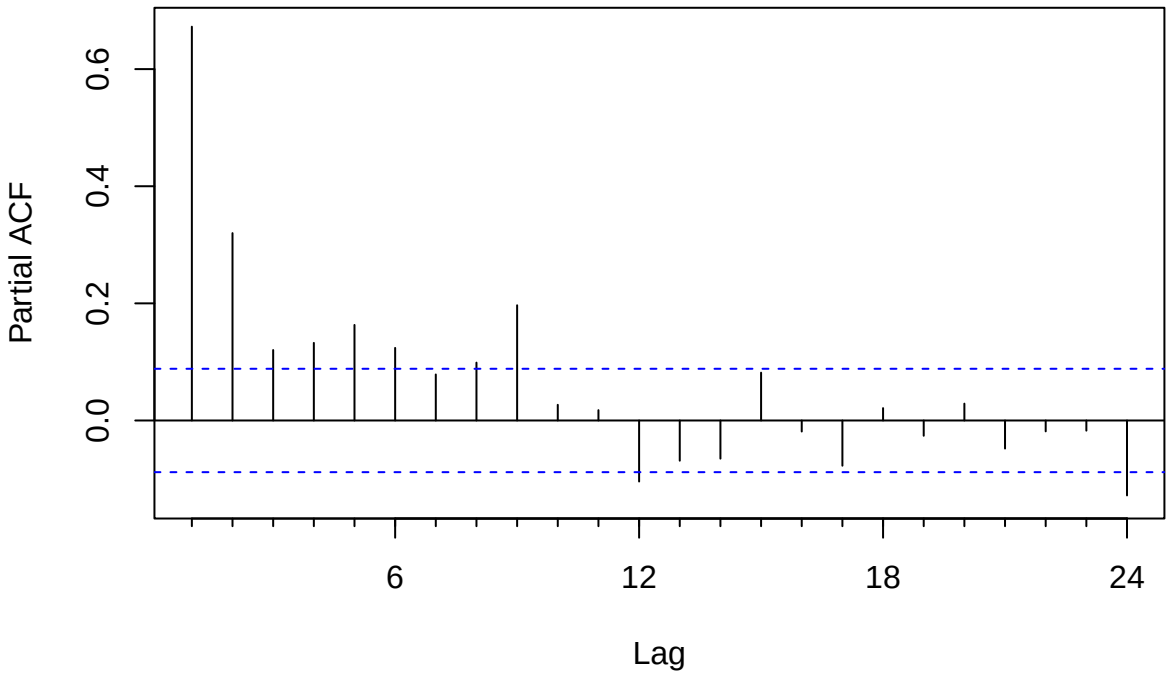


Now, we shall start the estimations

AR

In order to get some idea of what the order of our AR(p) process is, we plot the partial autocorrelation of the inflation series for a particular window.

```
# Partial autocorrelation
inflation %>%
    window(start = start(inflation), end = start(inflation) +
        c(0, 492)) %>%
    Pacf(lag.max = 24, plot = T)
```



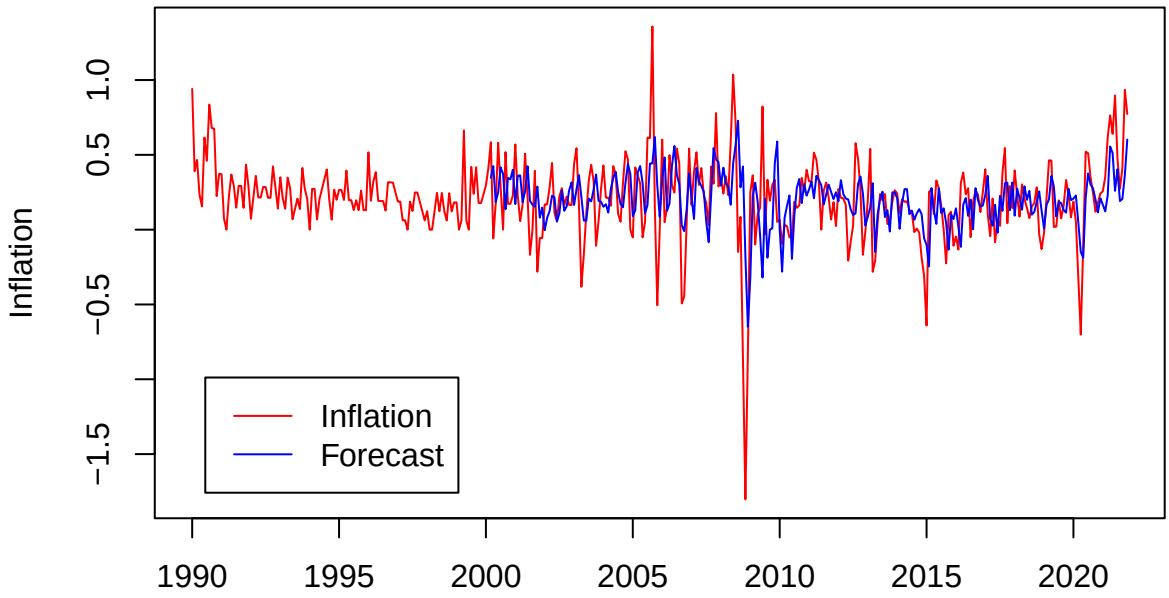
We believe that a maximum lag of 24 is more than reasonable. Then, the determine the actual order p based on the BIC.

```
# Function for calculating the BIC for AR models
BIC.ar <- function(model) {

    ssr <- sum(model$resid^2, na.rm = T)
    t <- length(model$resid)
    npar <- length(model$coef)

    return(c(p = model$order, BIC = log(ssr/t) + npar * log(t)/t))
}
```

We proceed to doing a rolling window one-step-ahead forecast, in which we choose the optimal order of the AR in each window of estimation.

```
# Rolling window forecasting
rolling_window <- 492
p.max <- 24

forecast1 = list()

for (a in 0:(length(inflation) - rolling_window - 1)) {

    # get the window for training the model and the one for
    # testing
    train = window(inflation, start = start(inflation) + c(0,
        a), end = start(inflation) + c(0, a + rolling_window -
        1))
    # test = window(inflation, start =
```

```
    # start(inflation)+c(0,a+rolling_window), end =
    # start(inflation)+c(0,a+rolling_window))

    bic.table = c()

    for (p in 0:p.max) {
        # calculating the BIC for different orders of the
        # AR(p)
        AR = ar(train, order.max = p, method = "ols", aic = F)
        bic.line = BIC.ar(AR)
        bic.table = rbind(bic.table, bic.line)
    }
    bic.table = data.frame(bic.table)

    p.opt = bic.table$p[which.min(bic.table$BIC)]  # pick the optimal p

    AR = ar(train, order.max = p.opt, method = "ols", aic = F)  # run the AR model with the optimal p

    forecast1[[a + 1]] = predict(AR, n.ahead = 1)$pred  # one-step-ahead forecast
}

forecasts = forecast1 %>%
    unlist() %>%
    ts(start = start(forecast1[[1]]), frequency = frequency(forecast1[[1]]))
```



AR + PC

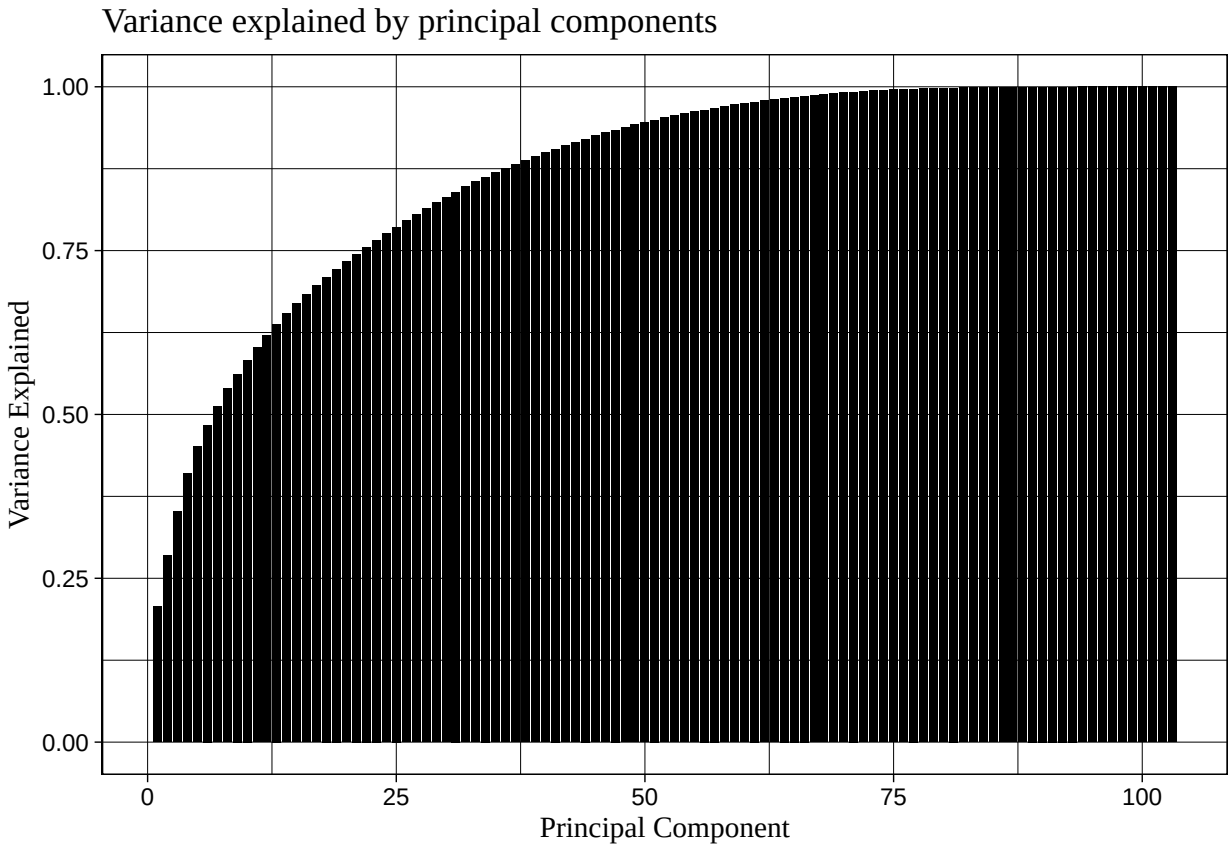
1. PCA

We do a Principal Component Analysis (PCA). Note that we must center and scale the data, since the series are in different scales.

```
# PCA
pca = data %>%
    select(-CPIAUCSL, -date) %>%
    prcomp(center = TRUE, scale = TRUE)
```

2. Select PCs

We can, then, choose the number of factors k and select the first k PCs. As seen in Question 1, there are different ways to choose the number of factors. We look at 3 common criterion (rule of thumb, informal way and biggest drop), but we opt for the rule of thumb as it seems to be the most parsimonious in this case. colocar alguma explicacao de qual criterio vamos adotar - usar rule of thumb q foi oq o masuko usou na 1




```
# Choosing the number of PCs

# Rule of thumb (3%)

pca.var.prop %>%
    filter(var.prop >= 0.03) %>%
    nrow() %>%
    paste("(rule of thumb)")
```

```
## [1] "6 (rule of thumb)"
```

```
# Informal way (90%)

pca.var.prop %>%
    filter(var.prop.cum <= 0.9) %>%
    nrow() %>%
    paste("(informal way)")
```

```
## [1] "40 (informal way)"
```

```r
# Biggest drop

(lag(pca.var.prop$var.prop)/pca.var.prop$var.prop) %>%
    which.max() %>%
    -1 %>%
    paste("(biggest drop)")
```

```
## [1] "102 (biggest drop)"
```

```r
# Using the rule of thumb
n_pc = pca.var.prop %>%
    filter(var.prop >= 0.03) %>%
    nrow()
```

3. Regression

Given the number of factors, the order of the autoregressive component is determined by BIC in each rolling window.

```r
# Get the factor from the PCA
Factors = pca$x[, 1:n_pc]

# Create the data matrix with the factors
variables = cbind(inflation, Factors)

variables_withdate = variables %>%
    bind_cols(date = as.Date.yearmon(time(inflation))) %>%
    setNames(c("inflation", colnames(Factors), "date"))
```

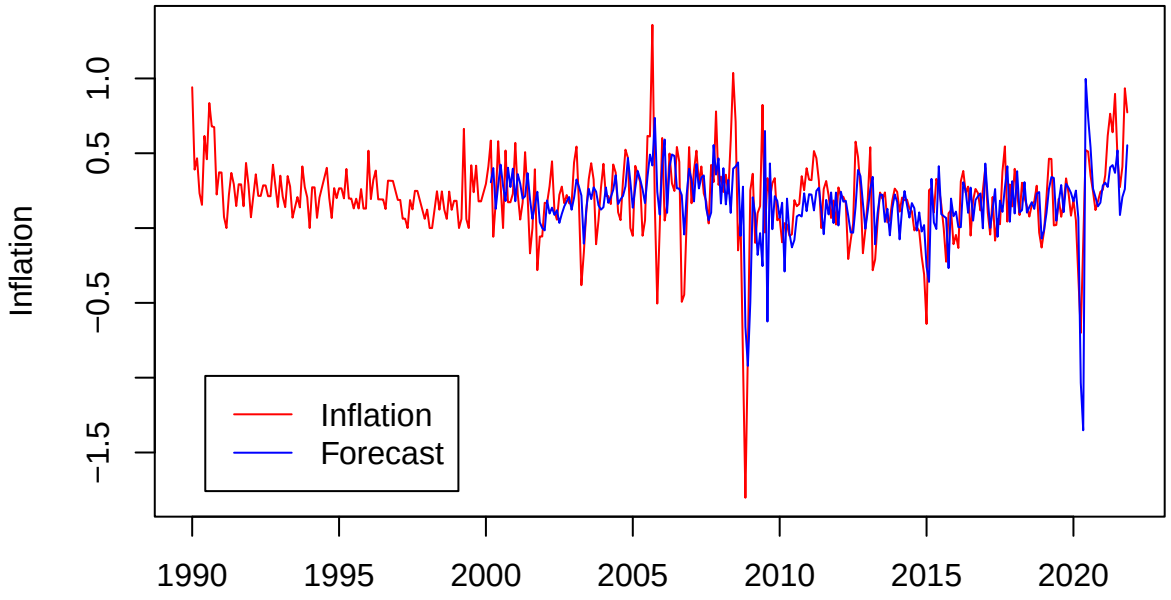
We proceed with the rolling window one-step-ahead forecast.

```r
# Function for creating the proper data matrix based on the
# regression formula instead of manually creating data
# matrix, we use the dynlm() function and get only the
# $model component
create_datamatrix = function(train, p.opt) {
    new = dynlm(inflation ~ L(inflation, 1:p.opt) + L(train[,
        -1], 1), data = ts(rbind(train, 0), start = start(train),
        frequency = frequency(train)))
    new = new$model %>%
        tail(1) %>%
        select(-inflation) %>%
        as.matrix()
    return(new)
}
```

paralelizar esse pedaco pra rodar mais rapido

```r
# Rolling window forecasting
rolling_window <- 492
p.max <- 24

forecast1 = list()

for (a in 0:(length(inflation) - rolling_window - 1)) {

    train = window(variables, start = start(inflation) + c(0,
```

```
        a), end = start(inflation) + c(0, a + rolling_window -
        1))
    test = window(variables, start = start(inflation) + c(0,
        a + rolling_window), end = start(inflation) + c(0, a +
        rolling_window))

    bic.table = rep(NA, p.max)

    for (p in 1:p.max) {
        # calculating the BIC for different orders of the
        # AR(p)
        AR_PC = dynlm(inflation ~ L(inflation, 1:p) + L(train[,
            -1], 1), data = train)
        bic.table[p] = BIC(AR_PC)
    }

    p.opt = which.min(bic.table)  # pick the optimal p

    AR_PC = dynlm(inflation ~ L(inflation, 1:p.opt) + L(train[,
        -1], 1), data = train)  # run the AR-PC model with the optimal p

    new = create_datamatrix(train, p.opt)

    forecast1[[a + 1]] = AR_PC$coefficients %*% c(1, as.matrix(new))  # one-step-ahead forecast
}

forecast1 = forecast1 %>%
    unlist() %>%
    ts(start = start(inflation) + c(0, rolling_window), frequency = frequency(inflation))
```

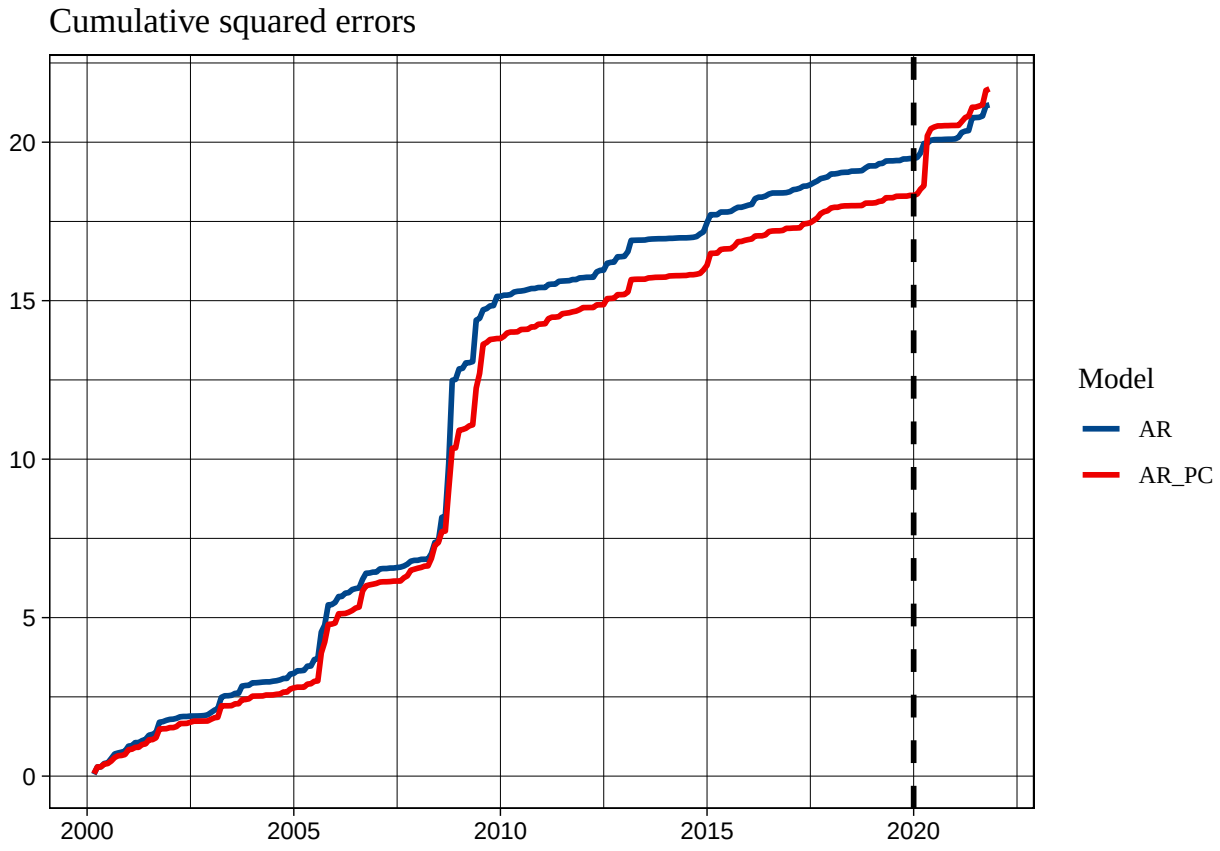


Item A

```
# Forecasting error
forecasts = cbind(AR = forecasts, AR_PC = forecast1) %>%
```

```
    as.ts()
error = inflation - forecasts
cum_error = sapply(error, function(x) {
    x^2 %>%
        cumsum()
}) %>%
    bind_cols(date = as.Date.yearmon(time(error))) %>%
    setNames(c("AR", "AR_PC", "date"))
```

```
# Plot
cum_error %>%
    gather(key = model, value = error, AR, AR_PC) %>%
    ggplot(aes(x = date, y = error, color = model)) + geom_line(size = 1) +
    geom_vline(xintercept = as.Date("2020-01-01"), linetype = "dashed",
        size = 1) + scale_color_lancet() + labs(title = "Cumulative squared errors",
    color = "Model", y = NULL, x = NULL)
```



Ridge Regression

LASSO Regression