Trabalho - Econometria IV

Guilherme Luz, Guilherme Masuko, Caio Garzeri

August 2023

```
library(lubridate)  # for handling dates
library(zoo)  # for time series
library(dynlm)  # for time series regressions
library(forecast)  # for the improved Pacf function
library(glmnet)  # for shrinkage methods
library(HDeconometrics)  # IC for glmnet

# Packages for parallel computation
library(future)
library(foreach)
library(doFuture)
library(doRNG)
```

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 $\pi_t = \frac{\Delta P_t}{P_{t-1}}$

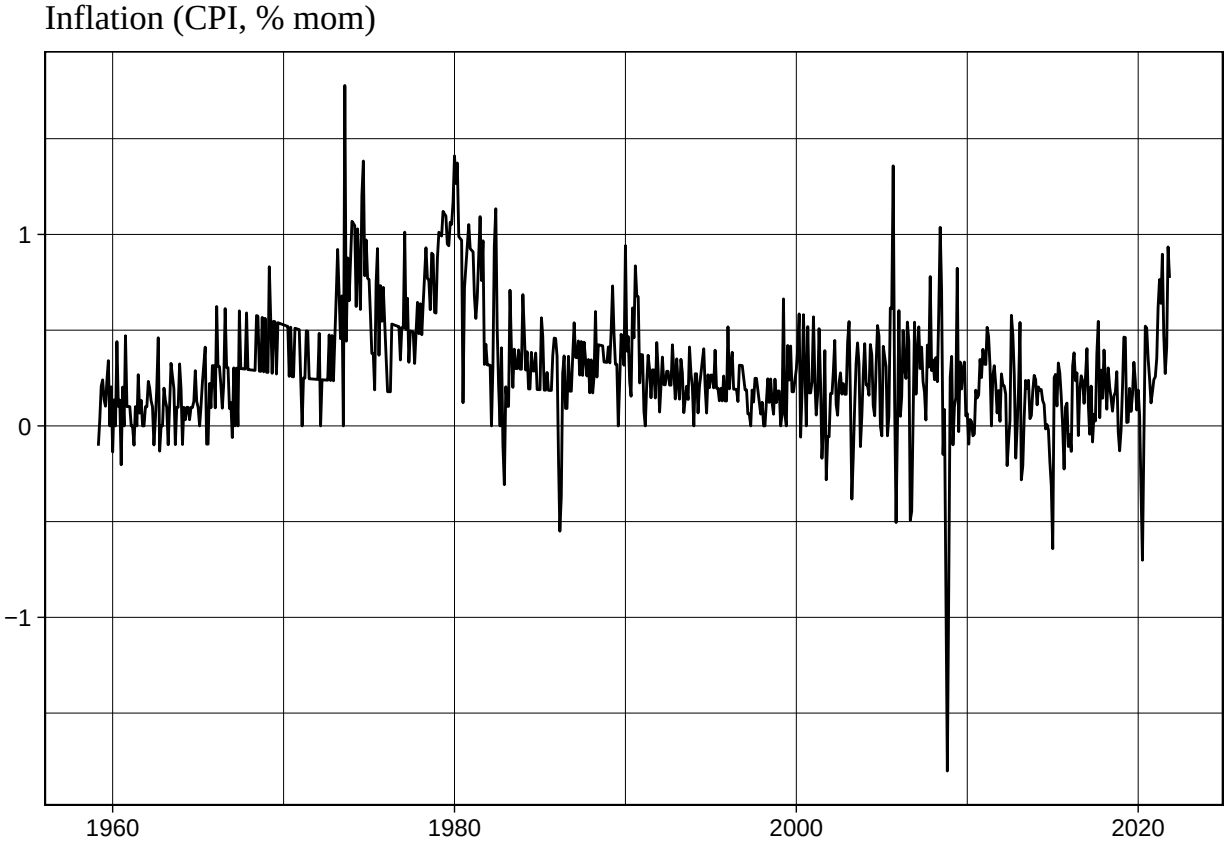
```
# Data transformations based on the FRED transformation
# codes
data = data0 %>%
    select(-sasdate) %>%
    rename(SP500 = "S&P 500", SPINDUST = "S&P: indust") %>%
```

```
    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)
```

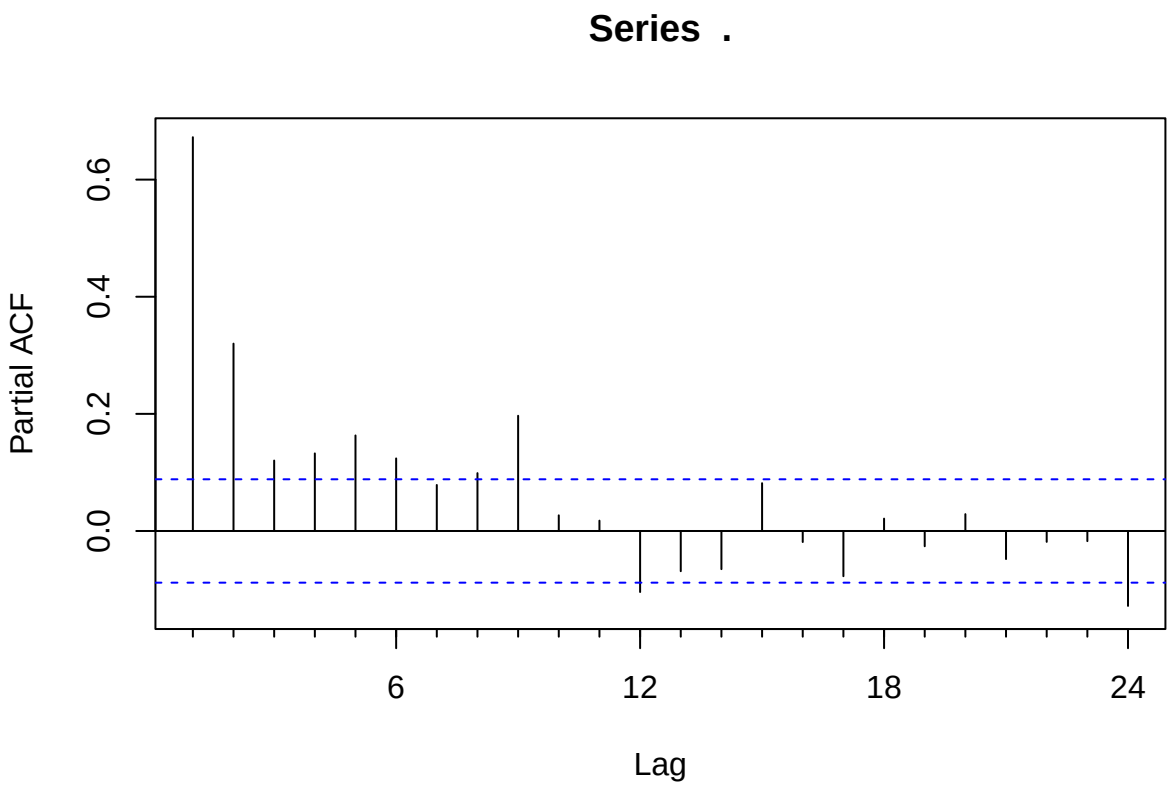


Now, we shall start the estimations.

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.

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



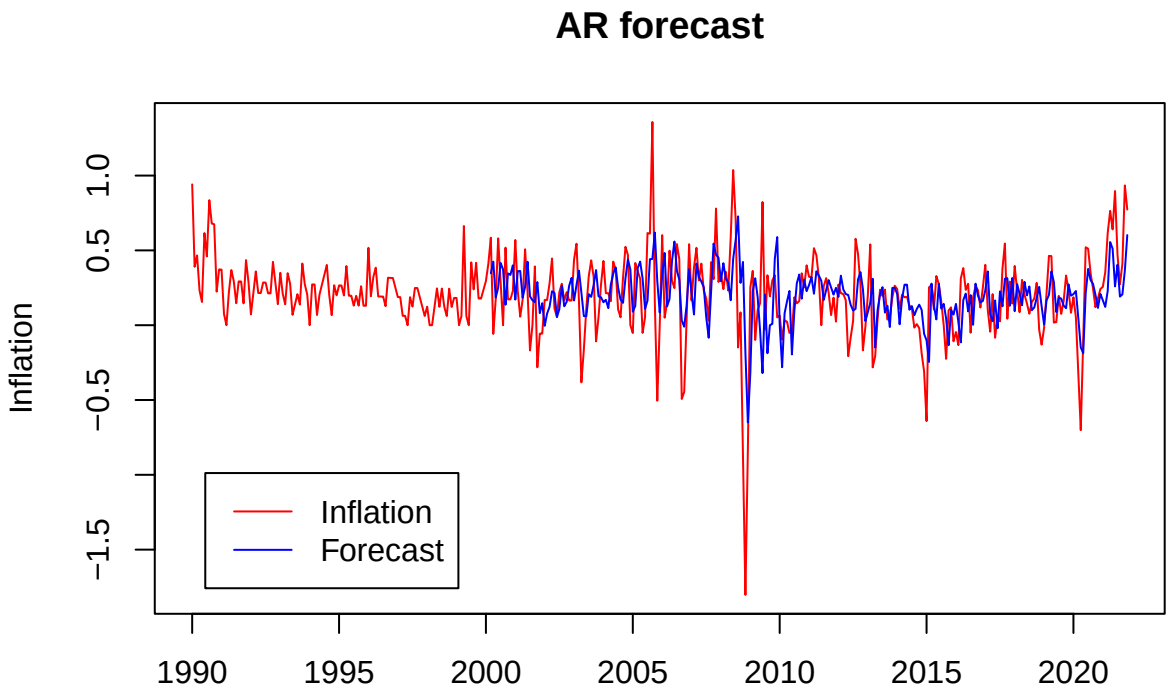
We believe that a maximum lag of 24 is more than reasonable. Then, the determine the actual order p based on the BIC.

```r
# 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 with a rolling window one-step-ahead forecast, in which we choose the optimal order of the AR in each window of estimation.

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

forecast1 = list()

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

```r
    # get the window for training the model
    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

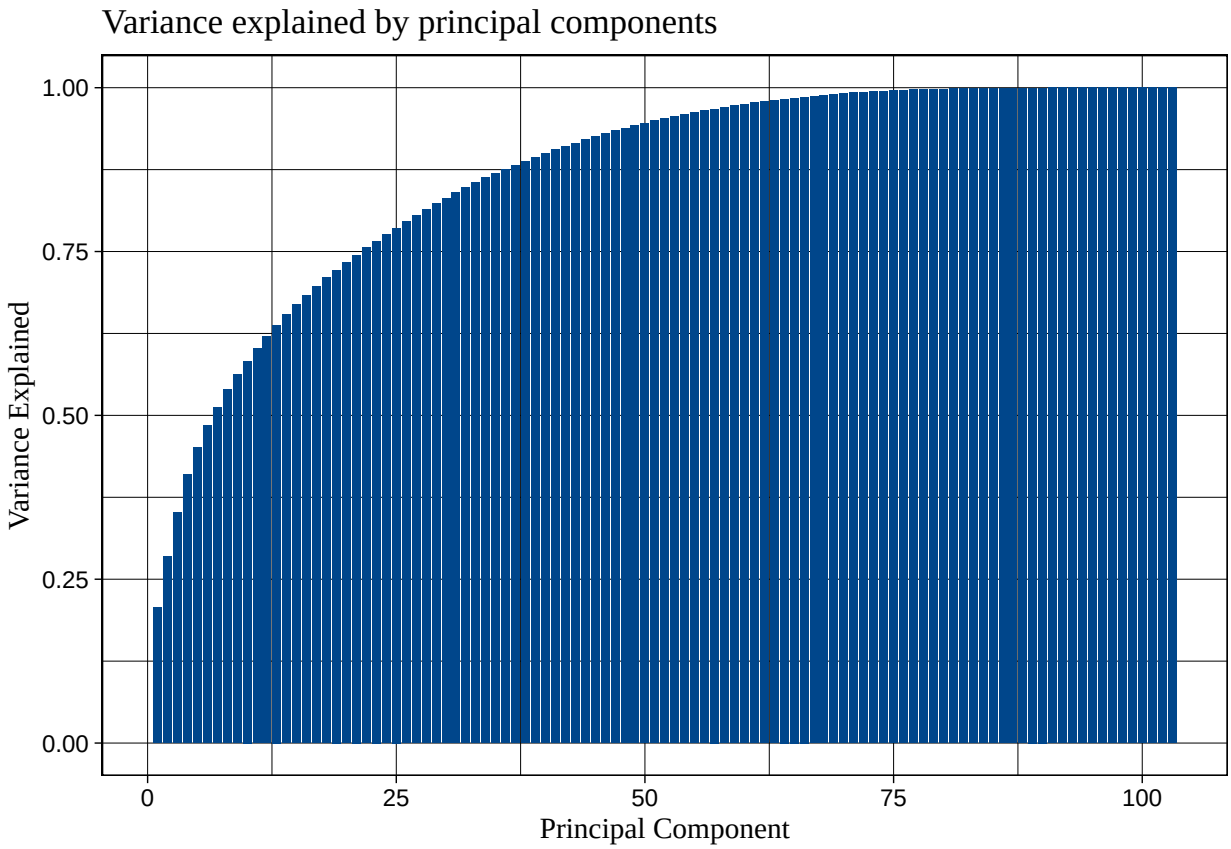
1. PCA

We do a Principal Component Analysis (PCA). Note that we must center and scale the data, since the series are in different scales. depois pra fazer previsao deveria me preocupar com isso?

```
# 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)"
```

```r
# 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.

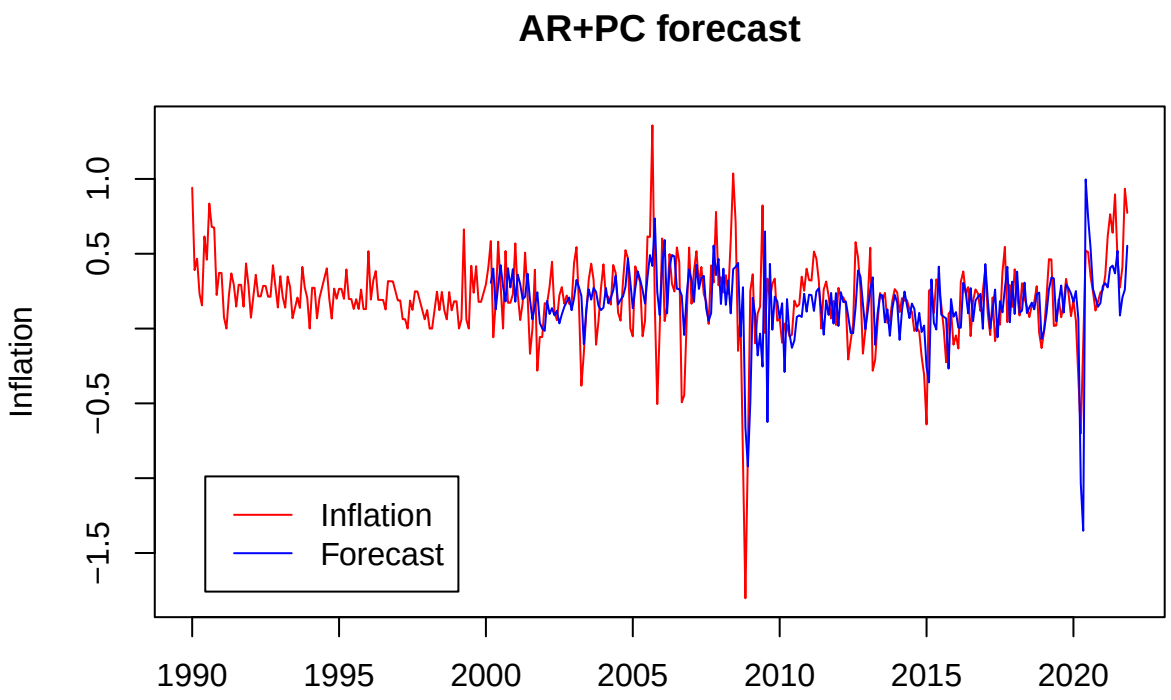
```r
# Get the factor from the PCA
Factors = pca$x[, 1:n_pc]

# 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; acho q nao precisa do test nesse pedaco tambem. Se nao precisar entao ja tira daqui e do chunk do AR

```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, new)  # one-step-ahead forecast
}

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




```
# Save forecasts
forecasts = cbind(AR = forecasts, AR_PC = forecast1) %>%
    as.ts()
```

Ridge Regression

We will choose penalty term according to the BIC. However, we must decide on the number of lags in the model and this criterion is obviously silent about this issue. Our strategy will be to run the models with 1, 2, 3 and 4 lags and choose the model with the smallest MSE.

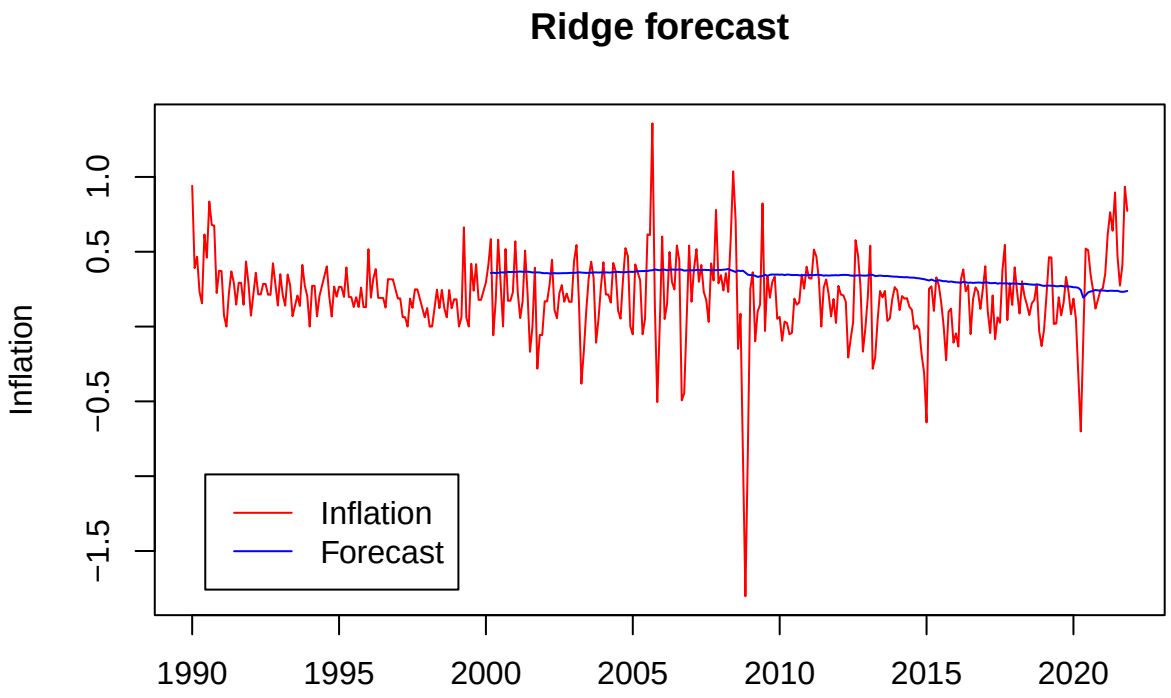
```
# Embedding function that creates n_lags of all variables
# of a given data frame
my_embed = function(df, n_lags = 4) {
    Lags = list()
    Lags[[1]] = df %>%
        select(-contains("date"))
    if (n_lags >= 1) {
        for (i in 1:n_lags) {
            Lags[[i + 1]] = df %>%
                select(-contains("date")) %>%
                mutate_all(function(x) lag(x, n = i))
        }
    }
    lagged_data = reduce(Lags, function(x, y) {
        bind_cols(x, y, .name_repair = ~make.unique(.x))
    })

    return(lagged_data)
}
```

```
tic()
# Rolling window forecasting
rolling_window <- 492
```

```r
# glmnet parameter
my_alpha = 0  # Ridge

forecast1 = list()

# set up parallel computation
registerDoFuture()
plan("multisession", workers = 3)  # use 3 cores

forecast1 = foreach(a = 1:(length(inflation) - rolling_window)) %dorng%
    {
        # get the window for training the model
        train = data[a:(a + rolling_window - 1), ]
        # embed
        reg_data = my_embed(train)
        # bind the embeded columns with the one-step-ahead
        # inflation
        reg_data = bind_cols(inflation.ahead = lead(inflation[a:(a +
            rolling_window - 1)]), reg_data)

        # Ridge estimation
        ic_ridge <- ic.glmnet(x = reg_data %>%
            na.omit() %>%
            select(-inflation.ahead), y = reg_data %>%
            na.omit() %>%
            select(inflation.ahead) %>%
            data.matrix(), crit = "bic", alpha = my_alpha)
        ridge <- glmnet(x = reg_data %>%
            na.omit() %>%
            select(-inflation.ahead), y = reg_data %>%
            na.omit() %>%
            select(inflation.ahead) %>%
            data.matrix(), alpha = my_alpha, lambda = ic_ridge$lambda)

        # Prediction
        new = reg_data %>%
            select(-inflation.ahead) %>%
            tail(1)
        result = predict(ridge, newx = data.matrix(new), s = ic_ridge$lambda)

        result
    }

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

```
## 1368.125 sec elapsed
```

```r
beepr::beep()
```



Using 4 lags of all variables

```
# Save forecasts
forecasts = cbind.zoo(forecasts, Ridge_4lags = forecast1) %>%
    as.ts()
```

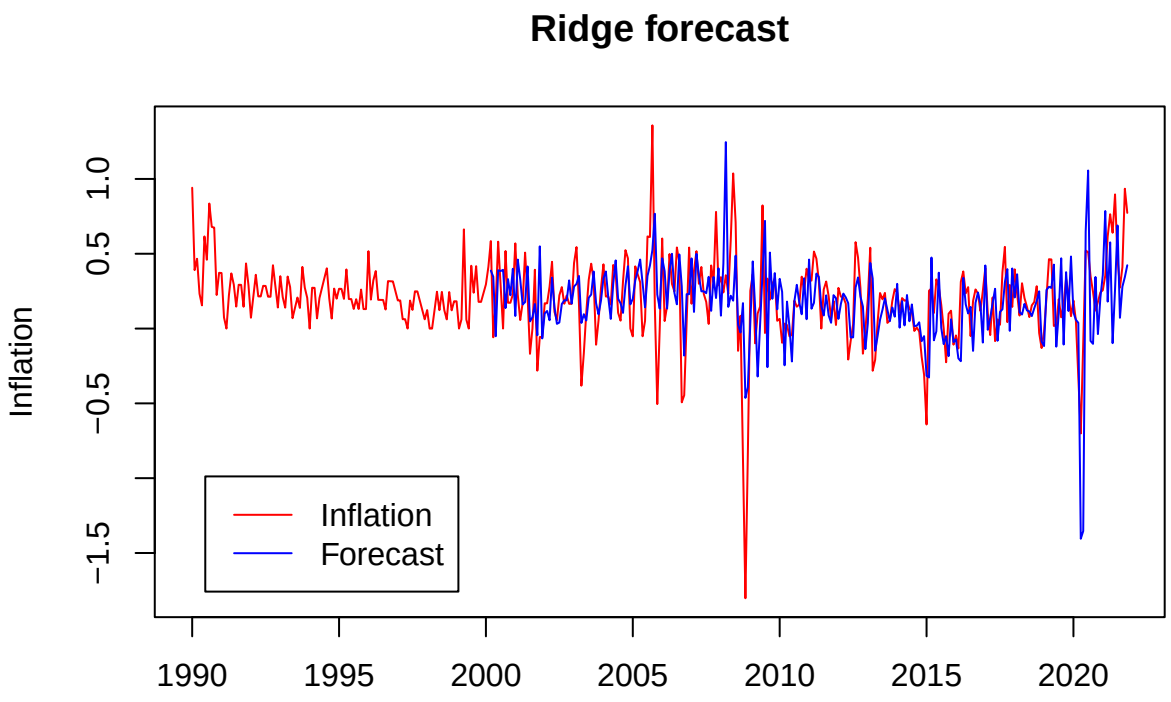
The forecast of the Ridge regression with 4 lags has a notably bad fit to the actual inflation series. We noticed that, since the ridge is not able to give a sparse solution, when there are too many variables, the estimated model becomes basically an intercept and almost all the other coefficients are very close to zero (but not zero). Hence, we tested other (more parsimonious) specifications. When we include the all the macroeconomics variables - without any lags - and lags of the CPI, we get a more reasonable result. The results are very robust to the number of CPI lags, so we keep 4 lags, as initially intended.

```
tic()
# Rolling window forecasting
rolling_window <- 492

# glmnet parameter
my_alpha = 0  # Ridge

forecast1 = list()

# set up parallel computation
registerDoFuture()
plan("multisession", workers = 3)  # use 3 cores

last_fcst = (length(inflation) - rolling_window)

forecast1 = foreach(a = 1:last_fcst) %dorng% {
    # get the window for training the model
    train = data[a:(a + rolling_window - 1), ] %>%
```

```
        select(-CPIAUCSL)
    train_cpi = data[a:(a + rolling_window - 1), ] %>%
        select(CPIAUCSL)
    # embed
    reg_data = my_embed(train, n_lags = 0)
    cpi_lags = my_embed(train_cpi, n_lags = 4)
    # bind the embeded columns with the one-step-ahead
    # inflation
    reg_data = bind_cols(inflation.ahead = lead(inflation[a:(a +
        rolling_window - 1)]), cpi_lags, reg_data)

    # Ridge estimation
    ic_ridge <- ic.glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), crit = "bic", alpha = my_alpha)
    ridge <- glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), alpha = my_alpha, lambda = ic_ridge$lambda)

    # Prediction
    new = reg_data %>%
        select(-inflation.ahead) %>%
        tail(1)
    result = predict(ridge, newx = data.matrix(new), s = ic_ridge$lambda)

    result
}

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

```
## 116.521 sec elapsed
```

```
beepr::beep()
```



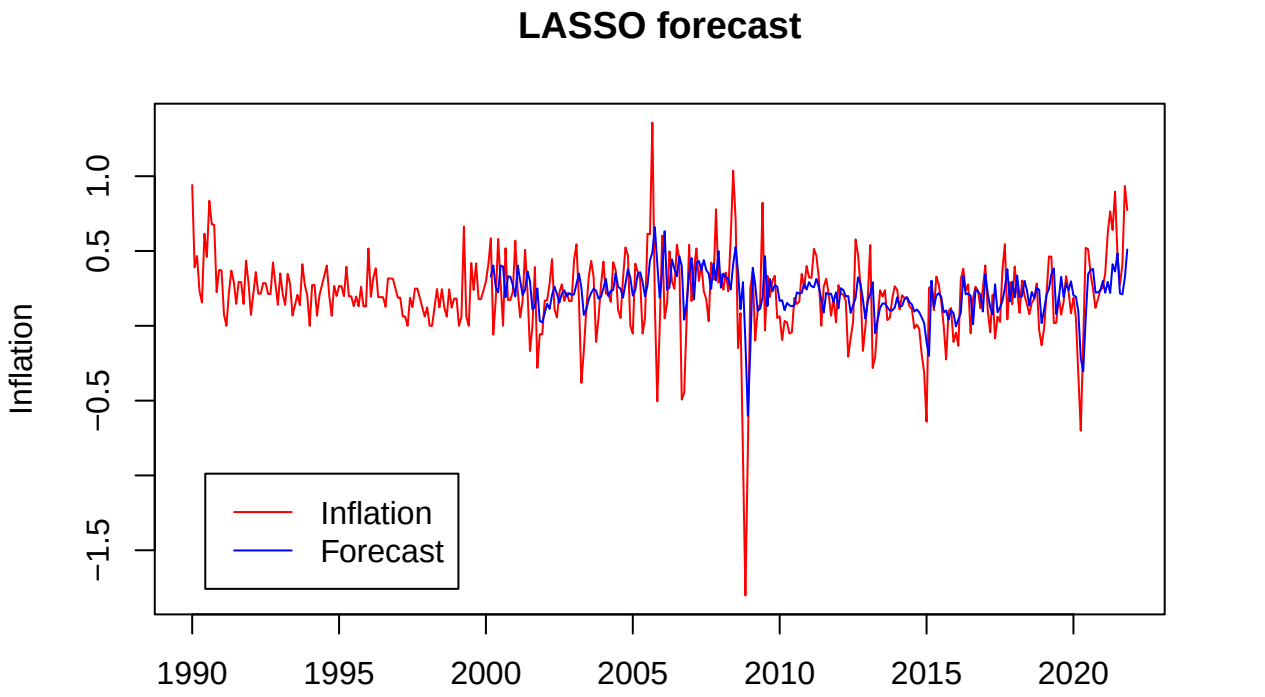
Using 4 lags of CPI and no lags of other variables

```
# Save forecasts
forecasts = cbind.zoo(forecasts, Ridge = forecast1) %>%
    as.ts()
```

LASSO Regression

```
# Rolling window forecasting
rolling_window <- 492

# glmnet parameter
my_alpha = 1  # LASSO

forecast1 = list()

for (a in 1:(length(inflation) - rolling_window)) {
    # get the window for training the model
    train = data[a:(a + rolling_window - 1), ]
    # embed
    reg_data = my_embed(train)
    # bind the embeded columns with the one-step-ahead
    # inflation
    reg_data = bind_cols(inflation.ahead = lead(inflation[a:(a +
        rolling_window - 1)]), reg_data)

    # Ridge estimation
    ic_lasso <- ic.glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
```

```
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), crit = "bic", alpha = my_alpha)
    lasso <- glmnet(x = reg_data %>%
        na.omit() %>%
        select(-inflation.ahead), y = reg_data %>%
        na.omit() %>%
        select(inflation.ahead) %>%
        data.matrix(), alpha = my_alpha, lambda = ic_lasso$lambda)

    # Prediction
    new = reg_data %>%
        select(-inflation.ahead) %>%
        tail(1)
    forecast1[a] = predict(lasso, newx = data.matrix(new), s = ic_lasso$lambda)
}

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

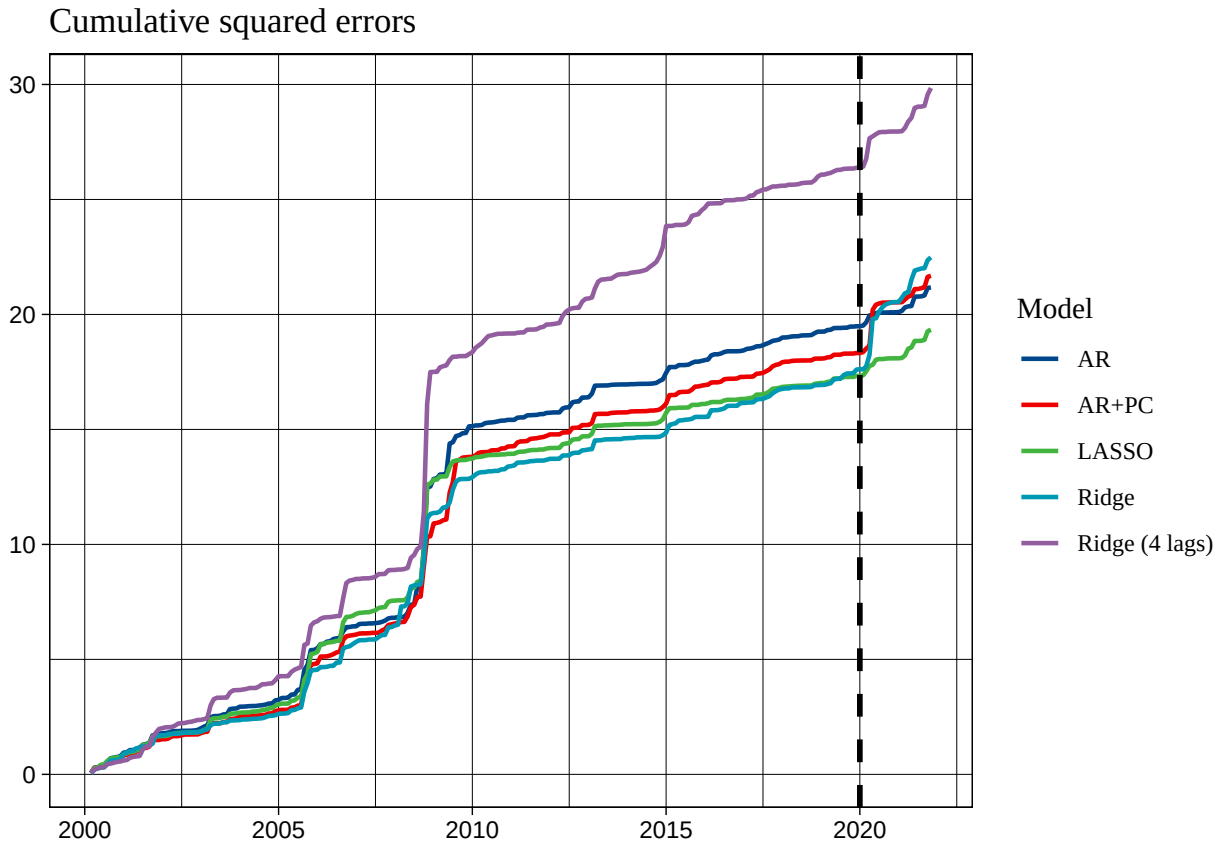



```
# Save forecasts
forecasts = cbind.zoo(forecasts, LASSO = forecast1) %>%
    as.ts()
```

Item A

```
# Forecasting error
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", "Ridge (4 lags)", "Ridge", "LASSO",
        "date"))
```




```
# Save data for Question 3
save(data, inflation, forecasts, file = "data/Q2_objects.Rda")
write.csv(forecasts, file = "output/forecasts.csv")
write.csv(cum_error, file = "output/cum_error.csv")
```