DFM Real Estate Prices Forecast

Pablo Barrio

2024-01-12

```
library(BVAR)
library(fbi)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'tibble'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
#functions locations
source("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Workspace/functions/remove_outliers.R")
source("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Workspace/functions/functions_diff_indices
source("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Workspace/functions/functions_3.R")
source("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Workspace/functions/transform_data.R")
source("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Project/functions_project.R")
```

```
## Import Data
# data is already transformed so that it is stationary
df_raw <- read.csv("C:/Users/pbarr/Documents/ENSAE/3A_MIE/S1/MacroECNM_ML/Project/fred_large_data.csv",

# Type of transformation performed on each series before factors are
DEMEAN <- 2 # --> demean and standardize

# Information criterion used to select the number of factors;
jj <- 2 #-> information criterion PC_p2

# Maximum number of factors to be estimated;
kmax <- 8
```

```
# ======================================================================
# PART 1: LOAD AND LABEL DATA
# We get rid of dates column
df <- df_raw[1:nrow(df_raw),2:length(df_raw)]

# Month/year of the final observation
final_date <- tail(df_raw$DATE, 1)
dates <- df_raw$DATE
```

```r
# T = number of months in the sample
TT <- length(dates)
```

```r
# ============================================================================
# PART 2: PROCESS DATA

class(df) <- c("data.frame", "fredmd")


##################################3

#select data
nn = 1 #corresponds to the index number of our target: CSUSHPISA in the following dataframe (HP in the

S_t = df[,c('CSUSHPISA', 'CURRCIR', 'PCE', 'TTLHHM156N', 'DPSACBW027SBOG', 'GDPr',
            'REAINTRATREARAT10Y', 'PSAVERT', 'MICH', 'CAPUTLG3311A2S', 'INDPRO',
            'IPB52300S', 'IPCONGD', 'IPDCONGD', 'IPG211S', 'IPG311A2S', 'IPG321S',
            'BOXRSA', 'CEXRSA', 'CHXRSA', 'DNXRSA', 'LXXRSA', 'MIXRSA', 'MNXRSA',
            'NYXRSA', 'PHXRSA', 'POXRSA', 'SDXRSA', 'SFXRSA', 'SPCS10RSA', 'TPXRSA',
            'WDXRSA', 'FLTOTALSL', 'NONREVSL', 'REVOLSL', 'TOTALSL',
            'COREFLEXCPIM159SFRBATL', 'CORESTICKM157SFRBATL',
            'CORESTICKM158SFRBATL', 'CORESTICKM159SFRBATL', 'CORESTICKM679SFRBATL',
            'CPIEALL', 'CPIEHOUSE', 'CWSR0000SA0', 'FLEXCPIM679SFRBATL',
            'IA001176M', 'IA001260M', 'MEDCPIM094SFRBCLE', 'MEDCPIM157SFRBCLE',
            'MEDCPIM158SFRBCLE', 'MEDCPIM159SFRBCLE', 'PCEPI', 'PCEPILFE',
            'PCETRIM12M159SFRBDAL', 'PCETRIM1M158SFRBDAL', 'PCETRIM6M680SFRBDAL',
            'STICKCPIM157SFRBATL', 'STICKCPIM159SFRBATL',
            'STICKCPIXSHLTRM159SFRBATL', 'TRMMEANCPIM158SFRBCLE', 'MSACSR',
            'BUSLOANS', 'CONSUMER', 'DPSACBM027SBOG', 'LOANINV', 'LOANS', 'REALLN',
            'TLAACBM027SBOG', 'USGSEC', 'CIVPART', 'LNS11300036', 'LNS11300060',
            'LNS11324230', 'M2REAL', 'M2SL', 'RMFSL', 'STDSL', 'LNS14000001',
            'LNS14000002', 'LNS14000024', 'LNS14000031', 'LNS14024887')]

X_t = df[,c('CSUSHPISA',
            "CPIAUCSL",
            "DSPIC96",
            "SPREAD",
            "MORTGAGE30US",
            "UNRATE")]



# We choose the horizon levels we want to forecast
HH = c(3,6)
```

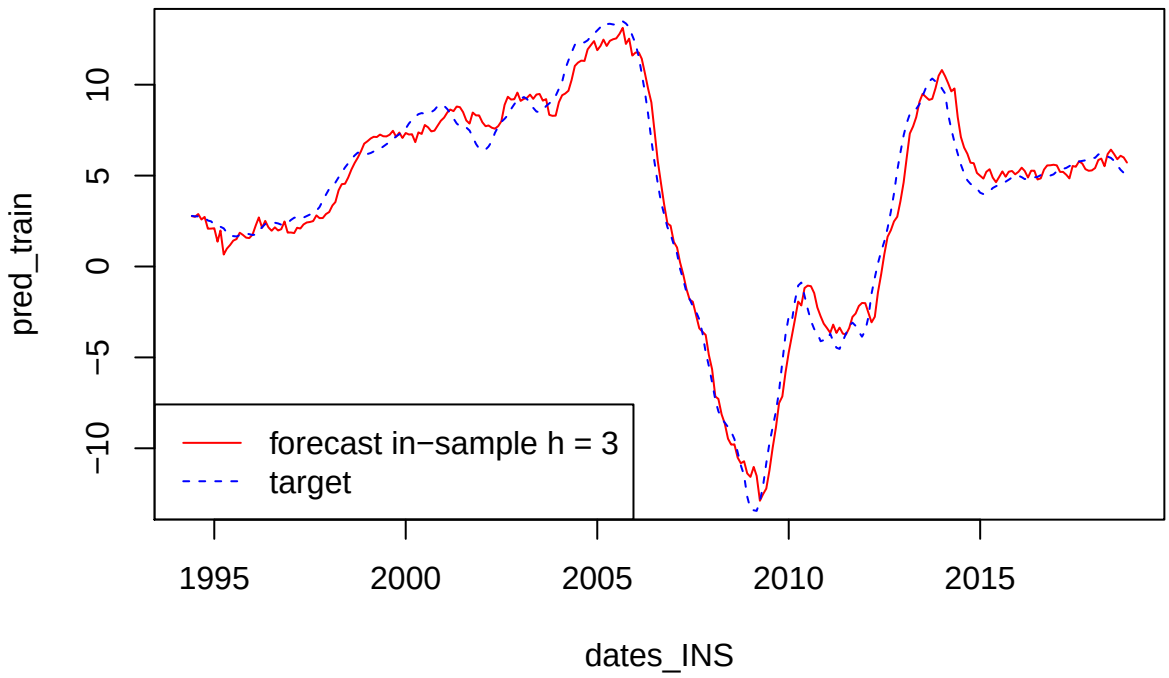
```r
# ============================================================================
### We define date related parameters for the out-of-sample and in-sample estimations

end_date = "2018-08-01"; #end date of in-sample
end_ins <- which(dates == end_date) #start index of out-of-sample
start_oos = end_ins + 1
```

```r
# Number of time points to use in the estimation of the parameter: Rolling scheme
wind_size = start_oos

j0 <- start_oos - wind_size + 1


# Prepare empty matrices that contain the results for out-of-sample
true <- matrix(NA, nrow = TT - tail(HH, 1) - start_oos +1, ncol = length(HH))
PC <- matrix(NA, nrow = TT - tail(HH, 1) - start_oos +1, ncol = length(HH)*3)
```

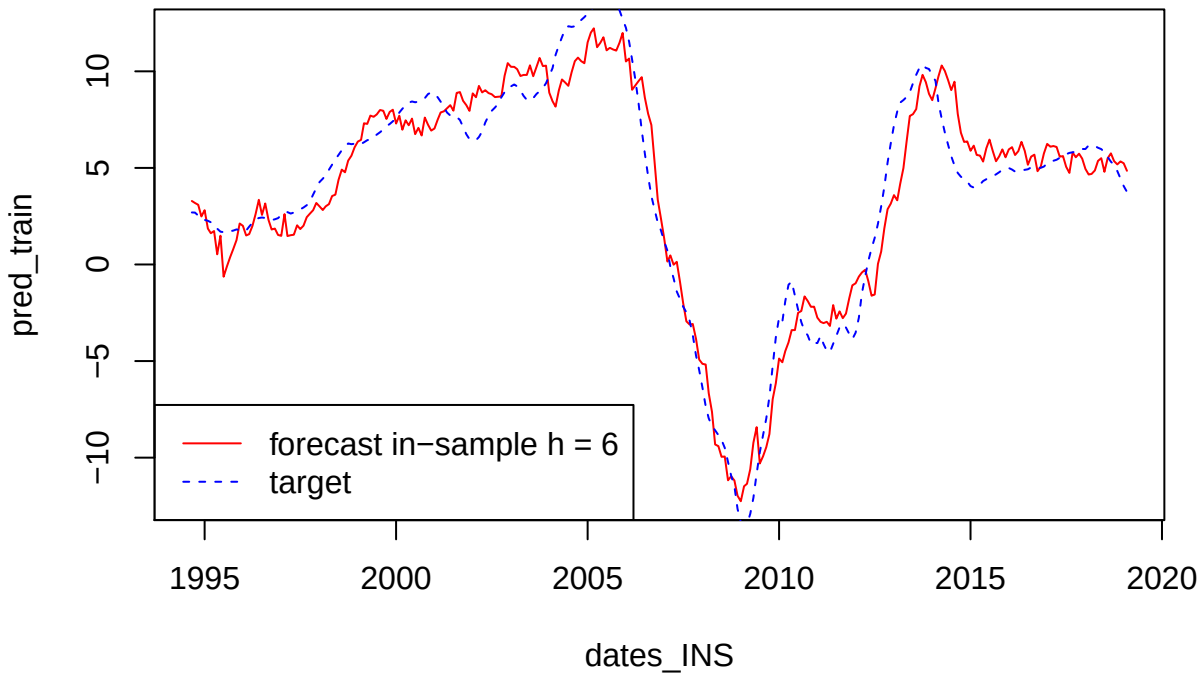
```r
############ 1. COMPUTE IN-SAMPLE

for (h in HH){

  S_temp1 <- remove_outliers(S_t[1:(end_ins+h), ])
  X_1 <- remove_outliers(X_t[1:end_ins, ])
  # We remove outliers from exogenous variables and also replace missing values by unconditionnal mean
  # Number of observations per series in x_new (i.e. number of rows)
  T <- nrow(X_1)

  # Get unconditional mean of the non-missing values of each series
  mut <- matrix(rep(colMeans(X_1, na.rm = TRUE), T), nrow = nrow(X_1), ncol = ncol(X_1), byrow = TRUE)

  # Replace missing values with unconditional mean
  X_1[is.na(X_1)] <- mut[is.na(X_1)]          # we replace the NA values in the vector X_1 with the corr
  # values from the vector mut.
  # Check whether there are entire columns of zeros
  Index_zeros <- which(colSums(X_1==0) == nrow(X_1))
  if ((length(Index_zeros))==0){
    X_1_new <- X_1
  } else {
    X_1_new <- subset(X_1,select = -Index_zeros)
  }
  # Demean and standardize data
  X_2 <- transform_data(X_1_new, DEMEAN)
  X_temp1 <- X_2$x22

  #Estimate factor on whole training period
  S_pred1 <- S_temp1[1:(end_ins), -1]
  result_factors2 <- factors_em_2(S_pred1, kmax, jj, DEMEAN)

  # Regressors
  Z <- as.matrix(cbind(X_temp1, result_factors2$Fhat)) #model with factors, X_t and y_t
  #Z <- as.matrix(cbind(X_temp1[,1], result_factors2$Fhat)) #model with factorsand y_t
  #Z <- as.matrix(result_factors2$Fhat) #model with only factors

  # Compute the dependent variable to be predicted
  Y <- S_temp1[,nn]
  my = mean(Y)
  sy = sd(Y)
  Y_std = (Y-my)/sy

  # Compute the forecasts
```

```
  Z_trimmed <- Z[1:(nrow(Z)), ]
  gamma <- solve(t(Z_trimmed) %*% Z_trimmed) %*% t(Z_trimmed) %*% Y_std[(h+1):length(Y_std)]
  pred_train <- (Z_trimmed %*% gamma)*sy + my

  # Plot the results in-sample
  dates_INS = as.Date(dates[(1+h):(end_ins+h)])
  plot(dates_INS,pred_train, type = 'l', col = "red")
  lines(dates_INS, S_temp1[(1+h):(end_ins+h),1], col="blue",lty=2)
  legend("bottomleft", legend = c(paste("forecast in-sample h =",h), "target"), col = c("red", "blue"),

}
```

```
## Iteration 1: obj 999.000000 IC 8
```




```
## Iteration 1: obj 999.000000 IC 8
```




```
for (j in start_oos:(TT - tail(HH, 1) )) {
  # Remark that TT - tail(HH, 1) is '2023-09-01'. This is the last period of the out-of-sample
  # for each j we forecast the target at date j+h

  ## Displays the dates at the beginning of each month
  cat('--------------\n')
  cat('now running\n')
  cat(paste(dates[j], collapse = ' '), '\n')


  ## Define the beginning of the estimation sample
  j0 <- j - wind_size + 1  # Starting period for the in-sample

  # We remove outliers from all variables

  S_temp <- remove_outliers(S_t[j0:j, ])
  X_1 <- remove_outliers(X_t[j0:j, ])

  # We remove outliers from Exogenous variables X_t and replace missing values by unconditionnal mean o
  # Number of observations per series in x_new (i.e. number of rows)
  T <- nrow(X_1)


  # Fill in missing values for each series with the unconditional mean of that series.
  # Demean and standardize the updated dataset. Estimate factors using the demeaned and standardized da
  # and use these factors to predict the original dataset.
```

```r
# Get unconditional mean of the non-missing values of each series
mut <- matrix(rep(colMeans(X_1, na.rm = TRUE), T), nrow = nrow(X_1), ncol = ncol(X_1), byrow = TRUE)

# Replace missing values with unconditional mean
X_2 <- X_1
X_2[is.na(X_2)] <- mut[is.na(X_2)] # we replace the NA values in the vector x2 with the corresponding

# We check whether there are entire columns of zeros
Index_zeros <- which(colSums(X_2==0) == nrow(X_2))
if ((length(Index_zeros))==0){
  X_2_new <- X_2
} else {
  X_2_new <- subset(X_2,select = -Index_zeros)
}


# We demean and standardize data
X_3 <- transform_data(X_2_new, DEMEAN)
X_temp <- X_3$x22

i = 0 # We use this as an index for each forecast horizon

for (h in HH) {  # Loop across the number of steps ahead

  i = i+1


  ## We keep the true value to be predicted
  true[j - wind_size + 1, i] <- S_t[ j+h, nn]

  ## We compute the factors
  result_factors <- factors_em_2(S_temp[,-nn], kmax, jj, DEMEAN)

  # Regressors
  A <- as.matrix(cbind(X_temp, result_factors$Fhat)) #model with factors, exogenous variables and act
  B <- as.matrix(cbind(X_temp[,1], result_factors$Fhat)) #model with factors and actual value of CSUS
  C <- as.matrix(result_factors$Fhat) #model with only the factors as regressors


  #We standardize the dependent variable to be predicted
  Y <- S_temp[,nn]
  my = mean(Y)
  sy = sd(Y)
  Y_std = (Y-my)/sy

  # Compute the forecasts by OLS for each model

  A_trimmed <- A[1:(nrow(A)-h), ]
  gamma_A <- solve(t(A_trimmed) %*% A_trimmed) %*% t(A_trimmed) %*% Y_std[(h+1):length(Y_std)]
  pred_A <- tail(A, 1) %*% gamma_A

  B_trimmed <- B[1:(nrow(B)-h), ]
  gamma_B <- solve(t(B_trimmed) %*% B_trimmed) %*% t(B_trimmed) %*% Y_std[(h+1):length(Y_std)]
```

```
    pred_B <- tail(B, 1) %*% gamma_B

    C_trimmed <- C[1:(nrow(C)-h), ]
    gamma_C <- solve(t(C_trimmed) %*% C_trimmed) %*% t(C_trimmed) %*% Y_std[(h+1):length(Y_std)]
    pred_C <- tail(C, 1) %*% gamma_C

    PC[j - wind_size + 1, i] <- (pred_A*sy + my)
    PC[j - wind_size + 1, i + 2] <- (pred_B*sy + my)
    PC[j - wind_size + 1, i + 4] <- (pred_C*sy + my)

  }
}
```

```
## We compute the RMSE and the R^2 for all  models at all horizons
# horizon 3 months
true_NA_3 <- na.omit(true[,1])
PC_NA_3_A <- na.omit(PC[,1])
PC_NA_3_B <- na.omit(PC[,3])
PC_NA_3_C <- na.omit(PC[,5])

#horizon 6 months
true_NA_6 <- na.omit(true[,2])
PC_NA_6_A <- na.omit(PC[,2])
PC_NA_6_B <- na.omit(PC[,4])
PC_NA_6_C <- na.omit(PC[,6])

RMSE_PC_3_A <- sqrt(mean((true_NA_3 - PC_NA_3_A)^2))
RMSE_PC_3_B <- sqrt(mean((true_NA_3 - PC_NA_3_B)^2))
RMSE_PC_3_C <- sqrt(mean((true_NA_3 - PC_NA_3_C)^2))
R_2_3_A <- calculate_adjusted_r_squared(true_NA_3, PC_NA_3_A, 14)
R_2_3_B <- calculate_adjusted_r_squared(true_NA_3, PC_NA_3_B, 9)
R_2_3_C <- calculate_adjusted_r_squared(true_NA_3, PC_NA_3_C, 8)


RMSE_PC_6_A <- sqrt(mean((true_NA_6 - PC_NA_6_A)^2))
RMSE_PC_6_B <- sqrt(mean((true_NA_6 - PC_NA_6_B)^2))
RMSE_PC_6_C <- sqrt(mean((true_NA_6 - PC_NA_6_C)^2))
R_2_6_A <- calculate_adjusted_r_squared(true_NA_6, PC_NA_6_A, 14)
R_2_6_B <- calculate_adjusted_r_squared(true_NA_6, PC_NA_6_B, 9)
R_2_6_C <- calculate_adjusted_r_squared(true_NA_6, PC_NA_6_C, 8)

# We plot the results
h = 3

dates_OOS <- as.Date(dates[(start_oos + h):(length(dates)-tail(HH, 1) +h)])
plot(dates_OOS, PC_NA_3_A, type = 'l', col = "red")
lines(dates_OOS, true_NA_3, col="blue",lty=2)
legend("bottomleft", legend = c("forecast out-of-sample h=3", "target"), col = c("red", "blue"), lty =
```

```
h = 6

dates_OOS <- as.Date(dates[(start_oos + h):(length(dates)-tail(HH, 1) +h)])
plot(dates_OOS, PC_NA_6_A, type = 'l', col = "red")
```

```
lines(dates_OOS, true_NA_6, col="blue",lty=2)
legend("bottomleft", legend = c("forecast out-of-sample h=6", "target"), col = c("red", "blue"), lty = c
```

Functions Used

```
factors_em_2 <- function(x, kmax, jj, DEMEAN) {
  # PART 1: CHECKS

  # Check that x is not missing values for an entire row
  if (any(rowSums(is.na(x)) == ncol(x))) {
    stop("Input x contains entire row of missing values.")
  }

  # Check that x is not missing values for an entire column
  if (any(colSums(is.na(x)) == nrow(x)) || any(colSums(is.na(x)) == (nrow(x)-1))) {
    #stop("Input x contains entire column of missing values.")
    cat("Input x contains entire column of missing values.")
    Index_missing_1 <- which(colSums(is.na(x)) == nrow(x))     # It counts how many 'NA' are in a column
                                                               # number of NA in a colum is equal to the
                                                               # then, the condition is met.
    Index_missing_2 <- which(colSums(is.na(x)) == (nrow(x)-1))
    Index_missing <- c(Index_missing_1,Index_missing_2)
    if ((length(Index_missing))==0){
      x_new <- x
    } else {
      x_new <- subset(x,select = -Index_missing)
    }
  } else {x_new <- x}

  # Check that kmax is an integer between 1 and the number of columns of x, or 99
  if (!((kmax <= ncol(x_new) && kmax >= 1 && floor(kmax) == kmax) || kmax == 99)) {
    stop("Input kmax is specified incorrectly.")
  }

  # Check that jj is one of 1, 2, 3
  if (!(jj %in% c(1, 2, 3))) {
    stop("Input jj is specified incorrectly.")
  }

  # Check that DEMEAN is one of 0, 1, 2, 3
  if (!(DEMEAN %in% 0:3)) {
    stop("Input DEMEAN is specified incorrectly.")
  }

  # PART 2: SETUP

  # Maximum number of iterations for the EM algorithm
  maxit <- 50

  # Number of observations per series in x_new (i.e. number of rows)
  T <- nrow(x_new)
```

```r
# Set error to arbitrarily high number
err <- 999

# Set iteration counter to 0
it <- 0

# Locate missing values in x_new
x1 <- is.na(x_new)

# PART 3: INITIALIZE EM ALGORITHM
# Fill in missing values for each series with the unconditional mean of that series.
# Demean and standardize the updated dataset. Estimate factors using the demeaned and standardized da
# and use these factors to predict the original dataset.

# Get unconditional mean of the non-missing values of each series
mut <- matrix(rep(colMeans(x_new, na.rm = TRUE), T), nrow = nrow(x_new), ncol = ncol(x_new), byrow = 

# Replace missing values with unconditional mean
x2 <- x_new
x2[is.na(x2)] <- mut[is.na(x2)]          # we replace the NA values in the vector x2 with the correspo
                                        # values from the vector mut.
# Check whether there are entire columns of zeros
Index_zeros <- which(colSums(x2==0) == nrow(x2))
if ((length(Index_zeros))==0){
  x2_new <- x2
  x1_new <- x1
} else {
  x2_new <- subset(x2,select = -Index_zeros)
  x1_new <- subset(x1,select = - Index_zeros)
  x_new <- subset(x_new,select = -Index_zeros)
}

# Number of series in x2_new (i.e. number of columns)
N <- ncol(x2_new)

# Demean and standardize data
x3 <- transform_data(x2_new, DEMEAN)

# If input 'kmax' is not set to 99, use subfunction baing() to determine
# the number of factors to estimate. Otherwise, set number of factors equal
# to 8
if (kmax != 99) {
  icstar <- baing(x3$x22, kmax, jj)$ic1
} else {
  icstar <- 8
}

# Run principal components on updated dataset
pc_result <- pc2(x3$x22, icstar)
chat0 <- pc_result$chat
Fhat <- pc_result$fhat
lamhat <- pc_result$lambda
ve2 <- pc_result$ss
```

```r
# PART 4: PERFORM EM ALGORITHM
# Update missing values using values predicted by the latest set of factors.
# Demean and standardize the updated dataset. Estimate a new set of factors using the updated dataset
# Repeat the process until the factor estimates do not change.

# Run while error is large and have yet to exceed maximum number of iterations
while (err > 0.001 && it < maxit) {
  # INCREASE ITERATION COUNTER
  it <- it + 1

  # Display iteration counter, error, and number of factors
  cat(sprintf('Iteration %d: obj %10f IC %d \n', it, err, icstar))

  # UPDATE MISSING VALUES
  for (t in 1:T) {
    for (j in 1:N) {
      if (x1_new[t, j] == 1) {
        x2_new[t, j] <- chat0[t, j] * x3$sdt[t, j] + x3$mut[t, j]
      } else {
        x2_new[t, j] <- x_new[t, j]
      }
    }
  }

  # ESTIMATE FACTORS
  # Demean and standardize the new data and recalculate mut and sdt using subfunction "transform_data
  x3 <- transform_data(x2_new, DEMEAN)
  X3_x22 <- as.matrix(x3$x22)
  if (any(colSums(is.na(x3$x22)) == nrow(x3$x22))) {
    cat("Input x3$x22 contains entire column of missing values.")
    Index_missing <- which(colSums(is.na(x3$x22)) == nrow(x3$x22))      # It counts how many 'NA' are
                                                                        # number of NA in a colum is
                                                                        # then, the condition is met.
    x3_x22_new <- subset(x3$x22,select = -Index_missing)
  } else {x3_x22_new <- x3$x22}

  # Determine number of factors to estimate for the new dataset using subfunction "baing()"
  # (or set to 8 if kmax equals 99)
  if (kmax != 99) {
    icstar <- baing(x3_x22_new, kmax, jj)$ic1
  } else {
    icstar <- 8
  }

  # Run principal components on the new dataset using subfunction "pc2()"
  #   chat   = values of x22 predicted by the factors
  #   Fhat   = factors scaled by (1/sqrt(N)) where N is the number of
  #            series
  #   lamhat = factor loadings scaled by number of series
  #   ve2    = eigenvalues of x3'*x3
  pc_result <- pc2(x3_x22_new, icstar)
  chat <- pc_result$chat
```

```
    # CALCULATE NEW ERROR VALUE
    # Caclulate difference between the predicted values of the new dataset and the predicted values of
    # dataset
    diff <- chat - chat0

    # The error value is equal to the sum of the squared differences between "chat" and "chat0" divided
    # of the squared values of "chat0"
    v1 <- as.vector(diff)
    v2 <- as.vector(chat0)
    err <- sum(v1^2) / sum(v2^2)

    # Set chat0 equal to the current chat
    chat0 <- chat
  }

  # Produce warning if maximum number of iterations is reached
  if (it == maxit) {
    warning('Maximum number of iterations reached in EM algorithm')
  }
  # Final Output:
  Fhat <- pc_result$fhat
  lamhat <- pc_result$lambda
  ve2 <- pc_result$ss

  # FINAL DIFFERENCE
  # Calculate the difference between the initial dataset and the values predicted by the final set of f
  ehat <- x_new - chat * x3$sdt - x3$mut


  return(list(ehat = ehat, Fhat = Fhat, lamhat = lamhat, ve2 = ve2, x2 = x2))
}

#-------------------------------------------------------------------------------------------------------
################################################
################## SUBFUNCTIONS ################
################################################

#########################
#### SUBFUNCTION pc2 ####
#########################
pc2 <- function(X, nfac) {

  # Number of series in X (i.e. number of columns)
  N <- ncol(X)

  # Singular value decomposition: X'*X = U*S*V'
  XX <- as.matrix(X)
  svd_result <- svd(t(XX) %*% XX)

  # Factor loadings scaled by sqrt(N)
  lambda <- svd_result$u[, 1:nfac] * sqrt(N)

  # Factors scaled by 1/sqrt(N) (note that lambda is scaled by sqrt(N))
```

```r
  fhat <- XX %*% lambda / N

  # Estimate initial dataset X using the factors (note that U'=inv(U))
  chat <- fhat %*% t(lambda)

  # Identify eigenvalues of X'*X
  ss <- diag(svd_result$d)

  # Return the results
  return(list(chat = chat, fhat = fhat, lambda = lambda, ss = ss))
}

#########################
## SUBFUNCTION minindc ##
#########################
minindc <- function(x) {
  apply(x, 2, which.min)
}

baing <- function(X, kmax, jj) {
  # ===========================================================================
  # DESCRIPTION
  # This function determines the number of factors to be selected for a given
  # dataset using one of three information criteria specified by the user.
  # The user also specifies the maximum number of factors to be selected.
  #
  # ---------------------------------------------------------------------------
  # INPUTS
  #           X       = dataset (one series per column)
  #           kmax    = an integer indicating the maximum number of factors
  #                     to be estimated
  #           jj      = an integer indicating the information criterion used
  #                     for selecting the number of factors; it can take on
  #                     the following values:
  #                           1 (information criterion PC_p1)
  #                           2 (information criterion PC_p2)
  #                           3 (information criterion PC_p3)
  #
  # OUTPUTS
  #           ic1     = number of factors selected
  #           chat    = values of X predicted by the factors
  #           Fhat    = factors
  #           eigval  = eivenvalues of X'*X (or X*X' if N>T)
  # ---------------------------------------------------------------------------
  #
  # PART 1: SETUP

  # Number of observations per series (i.e. number of rows)
  T <- nrow(X)

  # Number of series (i.e. number of columns)
  N <- ncol(X)
```

```r
# Total number of observations
NT <- N * T

# Number of rows + columns
NT1 <- N + T

# PART 2: OVERFITTING PENALTY
# Determine penalty for overfitting based on the selected information criterion.

# Allocate memory for overfitting penalty
#CT <- numeric(kmax)

# Array containing possible number of factors that can be selected (1 to kmax)
ii <- 1:kmax

# The smaller of N and T
GCT <- min(N, T)

# Calculate penalty based on criterion determined by jj.
if (jj == 1){
  # Criterion PC_p1
  CT <- log(NT / NT1) * ii * NT1 / NT
} else if (jj == 2){
  # Criterion PC_p2
  CT <- (NT1 / NT) * log(min(N, T)) * ii
} else if (jj == 3){
  # Criterion PC_p3
  CT <- ii * log(GCT) / GCT
}

# PART 3: SELECT NUMBER OF FACTORS
# Perform principal component analysis on the dataset and select the number
# of factors that minimizes the specified information criterion.

# RUN PRINCIPAL COMPONENT ANALYSIS

# Get components, loadings, and eigenvalues
XX <-as.matrix(X)
if (T < N) {
  # Singular value decomposition
  svd_result <- svd(XX %*% t(XX))

  # Components
  Fhat0 <- sqrt(T) * svd_result$u

  # Loadings
  Lambda0 <- t(XX) %*% Fhat0 / T

} else {
  # Singular value decomposition
  svd_result <- svd(t(XX) %*% XX)       # Alternatively, you can use "eigen(t(XX) %*% XX)" and then "

  # Loadings
```

```r
  Lambda0 <- sqrt(N) * svd_result$u

  # Components
  Fhat0 <- XX %*% Lambda0 / N

}

# SELECT NUMBER OF FACTORS

# Preallocate memory
Sigma <- numeric(kmax + 1)  # sum of squared residuals divided by NT
# "numeric()": it creates numeric vectors or matrices.
#It initializes the vector or matrix with numeric values (defaulting to zeros).
IC1 <- matrix(0, nrow = 1, ncol = kmax + 1)  # information criterion value

# Loop through all possibilities for the number of factors
for (i in kmax:1) {

  # Identify factors as first i components
  Fhat <- Fhat0[, 1:i]

  # Identify factor loadings as first i loadings
  lambda <- Lambda0[, 1:i]

  # Predict X using i factors
  chat <- Fhat %*% t(lambda)

  # Residuals from predicting X using the factors
  ehat <- X - chat

  # Sum of squared residuals divided by NT
  Sigma[i] <- mean(colSums(ehat^2/T))

  # Value of the information criterion when using i factors
  IC1[i] <- log(Sigma[i]) + CT[i]
}

# Sum of squared residuals when using no factors to predict X (i.e.
# fitted values are set to 0)
Sigma[kmax + 1] <- mean(colSums(X^2) / T)

# Value of the information criterion when using no factors
IC1[, kmax + 1] <- log(Sigma[kmax + 1])

# Number of factors that minimizes the information criterion
ic1 <- minindc(t(IC1))

# Set ic1 = 0 if ic1 > kmax (i.e. no factors are selected if the value of the
# information criterion is minimized when no factors are used)
ic1 <- ifelse(ic1 > kmax, 0, ic1)

# PART 4: SAVE OTHER OUTPUT
```

```r
  # Factors and loadings when the number of factors set to kmax
  Fhat <- Fhat0[, 1:kmax]  # factors
  Lambda <- Lambda0[, 1:kmax]  # factor loadings

  # Predict X using kmax factors
  chat <- Fhat %*% t(Lambda)

  # Get the eigenvalues corresponding to X'*X (or X*X' if N > T)
  eigval <- svd_result$d

  # Return the results
  return(list(ic1 = ic1, chat = chat, Fhat = Fhat, eigval = eigval))
}


#########################
## SUBFUNCTION minindc ##
#########################
minindc <- function(x) {

  # DESCRIPTION
  # This function finds the index of the minimum value for each column of a given matrix. The function
  # the minimum value of each column occurs only once within that column. The function returns an error
  # not the case.
  apply(x, 2, which.min)    # "2": it specifies that the function ("which.min" in this case) should be
  # each column. If it were 1, the function would be applied to each row.
  # "which.min": it returns the index of the first minimum value in a vector.
}
mrsq <- function(Fhat, lamhat, ve2, series) {
  # =========================================================================
  # DESCRIPTION
  # This function computes the R-squared and marginal R-squared from
  # estimated factors and factor loadings.
  #
  # -------------------------------------------------------------------------
  # INPUTS
  #           Fhat    = estimated factors (one factor per column)
  #           lamhat  = factor loadings (one factor per column)
  #           ve2     = eigenvalues of covariance matrix
  #           series  = series names
  #
  # OUTPUTS
  #           R2      = R-squared for each series for each factor
  #           mR2     = marginal R-squared for each series for each factor
  #           mR2_F   = marginal R-squared for each factor
  #           R2_T    = total variation explained by all factors
  #           t10_s   = top 10 series that load most heavily on each factor
  #           t10_mR2 = marginal R-squared corresponding to top 10 series
  #                     that load most heavily on each factor
  #
  # N = number of series, ic = number of factors
  N <- nrow(lamhat)
  ic <- ncol(lamhat)
```

```r
  # Preallocate memory for output
  R2 <- matrix(NA, nrow = N, ncol = ic)
  mR2 <- matrix(NA, nrow = N, ncol = ic)
  t10_s <- matrix(NA, nrow = 10, ncol = ic)
  t10_mR2 <- matrix(NA, nrow = 10, ncol = ic)

  # Compute R-squared and marginal R-squared for each series for each factor
  for (i in 1:ic) {
    R2[, i] <- t(apply(Fhat[, 1:i] %*% t(lamhat[, 1:i]), 2, var))
    mR2[, i] <- apply(Fhat[, i] %*% t(lamhat[, i]), 2, var)
  }

  # Compute marginal R-squared for each factor
  mR2_F <- ve2 / sum(ve2)
  mR2_F <- mR2_F[1:ic]

  # Compute total variation explained by all factors
  R2_T <- sum(mR2_F)

  # Sort series by marginal R-squared for each factor
  ind <- apply(mR2, 2, order, decreasing = TRUE)

  # Get top 10 series that load most heavily on each factor and the
  # corresponding marginal R-squared values
  for (i in 1:ic) {
    t10_s[, i] <- series[ind[1:10, i]]
    t10_mR2[, i] <- mR2[ind[1:10, i], i]
  }

  return(list(R2 = R2, mR2 = mR2, mR2_F = mR2_F, R2_T = R2_T, t10_s = t10_s, t10_mR2 = t10_mR2))
}

# This functions allows to compute the adjusted R^2
calculate_adjusted_r_squared <- function(y_true, y_pred, k) {
  # y_true is a vector with the target
  # y_pred is the predicted vector
  # k corresponds to the number of regressors

  # Calculate the residuals
  residuals <- y_true - y_pred

  # Calculate the sum of squares
  ss_residuals <- sum(residuals^2)
  ss_total <- sum((y_true - mean(y_true))^2)

  # Calculate the number of observations
  n <- length(y_true)

  # Calculate Adjusted R-squared
  adjusted_r_squared <- 1 - (ss_residuals / ss_total)*((n - 1) / (n - k - 1))

  return(adjusted_r_squared)
}
```

```r
remove_outliers <- function(X) {
  # ========================================================================
  # DESCRIPTION:
  # This function takes a set of series aligned in the columns of a matrix
  # and replaces outliers with the value NA.
  #
  # ------------------------------------------------------------------------
  # INPUT:
  #           X   = dataset (one series per column)
  #
  # OUTPUT:
  #           Y   = dataset with outliers replaced with NA
  #
  # ------------------------------------------------------------------------
  # NOTES:
  #           1) Outlier definition: a data point x of a series X(:,i) is
  #           considered an outlier if abs(x-median)>10*interquartile_range.
  #
  #           2) This function ignores values of NaN and thus is capable of
  #           replacing outliers for series that have missing values.
  #
  # ========================================================================
  # Error checking
  if (!inherits(X, "fredmd"))
    stop("Object must be of class 'fredmd'")
  #             - inherits(object, "fredmd"): The inherits function is used to check if an object inher
  #               particular class.
  #             - !inherits(object, "fredmd"): The ! operator negates the result of the inherits functi
  #               condition is true if the object does not inherit from the class "fredmd".
  #             - stop("Object must be of class 'fredmd'"): If the condition is true, the stop function
  #               The stop function is used to generate an error message and halt the execution of the
  #             So, we are checking if the object is not of class "fredmd" and, if so, it stops the exe
  # ========================================================================
  # Calculate median of each series
  #             We use the 'apply' function to calculate the median for each
  #             column (i.e., margin 2) of a matrix 'X'
  #             -'X': This is the matrix or data frame for which you want to calculate column-wise medi
  #             -'2': This argument specifies that the operation should be applied to each column.
  #                 In R, 1 would mean rows, and 2 means columns.
  #             - stats::median: This is the function that will be applied to each column (or row) of t
  #               In this case, it's the median function from the stats package.
  #             - na.rm = TRUE: This argument specifies whether to remove missing values (NA) before ap
  #               the function. In this case, na.rm = TRUE means that any missing values in each column
  #               ignored when calculating the median.
  #  So, overall, this command is calculating the median of each column in the matrix X, ignoring any m
  #  in each column. The result will be a vector of median values, one for each column of the matrix.
  median_X <- apply(X, 2, stats::median, na.rm = TRUE)
  # median_X <- matrix(median_X)
  # median_X <- t(median_X)

  # Repeat median of each series over all data points in the series
  median_X_mat <- matrix(rep(median_X, nrow(X)), nrow = nrow(X),
                         ncol = ncol(X), byrow = TRUE)
```

```
  #             - rep(median_X, nrow(X)): The rep function is used to replicate the value median_X a ce
  #               of times. In this case, it is replicated nrow(X) times, where nrow(X) is the number o
  #               matrix X. This creates a vector of repeated median values.
  #             - matrix(...): This function is then used to convert the repeated median vector into a
  #                 nrow = nrow(X): Specifies the number of rows in the resulting matrix, which is the
  #                                 number of rows in the original matrix X.
  #                 ncol = ncol(X): Specifies the number of columns in the resulting matrix, which is t
  #                                 the number of columns in the original matrix X.
  #                 byrow = TRUE: This argument specifies that the values in the repeated median vector
  #                               filled into the matrix by rows. If byrow = FALSE (or not specified),
  # So, overall, this command creates a matrix where each row contains the same repeated median value,
  # number of rows and columns in the new matrix is the same as the number of rows and columns in the o
  # matrix.

  # Calculate quartiles
  Q <- apply(X, 2, stats::quantile, probs = c(0.25, 0.75), na.rm = TRUE)

  # Calculate interquartile range (IQR) of each series
  IQR <- Q[2,] - Q[1,]

  # Repeat IQR of each series over all data points in the series
  IQR_mat <- matrix(rep(IQR, nrow(X)), nrow = nrow(X),
                    ncol = ncol(X), byrow = TRUE)

  # Determine outliers
  Z <- abs(X - median_X_mat)
  outlier <- (Z > (10 * IQR_mat))

  # Replace outliers with NaN
  Y <- X
  Y[outlier] <- NA

  # Cleaned data
  outdata <- Y
  # We set the class attribute of the object outdata to be a combination of two classes: "data.frame" a
  # In R, an object can belong to multiple classes. The class function is used to query or set the clas
  # In this case, it's setting the class of outdata to be both "data.frame" and "fredmd".
  class(outdata) <- c("data.frame", "fredmd")
  return(outdata)

  # Print the number of outliers
  print("Number of outliers:", quote = FALSE)     # quote = FALSE: The quote argument in the print func
  # whether or not to print quotation marks around the character
  # string. When quote = FALSE, it means that the string will be
  # printed without quotation marks.
  print(sum(outlier, na.rm = TRUE), quote = FALSE)  # na.rm = TRUE: The na.rm argument is set to TRUE,
  # that any missing values (NA) in the vector or matrix will be
  # removed before calculating the sum.
  # If na.rm is set to FALSE or not specified, the presence of any
  # missing values would result in the sum being reported as NA.


}
```

```r
transform_data <- function(x2, DEMEAN) {

  # ========================================================================
  # DESCRIPTION
  # This function transforms a given set of series based upon the input variable DEMEAN.
  #  The following transformations are possible:
  #
  #   1) No transformation.
  #
  #   2) Each series is demeaned only (i.e. each series is rescaled to have a mean of 0).
  #
  #   3) Each series is demeaned and standardized (i.e. each series is rescaled to have a mean of 0 and
  #       deviation of 1).
  #
  #   4) Each series is recursively demeaned and then standardized. For a given series x(t), where t=1,
  #     the recursively demeaned series x'(t) is calculated as x'(t) = x(t) - mean(x(1:t)). After the r
  #     demeaned series x'(t) is calculated, it is standardized by dividing x'(t) by the standard devia
  #     original series x. Note that this transformation does not rescale the original series to have a
  #     or standard deviation.
  #
  # ------------------------------------------------------------------------
  # INPUTS
  #           x2 = set of series to be transformed (one series per column); no missing values;
  #           DEMEAN  = an integer indicating the type of transformation performed on each series in x2
  #                     on the following values:
  #                           0 (no transformation)
  #                           1 (demean only)
  #                           2 (demean and standardize)
  #                           3 (recursively demean and then standardize)
  #
  # OUTPUTS
  #           x22     = transformed dataset
  #           mut     = matrix containing the values subtracted from x2
  #                     during the transformation
  #           sdt     = matrix containing the values that x2 was divided by
  #                     during the transformation
  #
  # ========================================================================
  # FUNCTION
  # Number of observations in each series (i.e. number of rows in x2)
  T <- nrow(x2)

  # Number of series (i.e. number of columns in x2)
  N <- ncol(x2)

  # Perform transformation based on type determined by 'DEMEAN'
  if (DEMEAN == 0){
    # No transformation
    mut <- matrix(0, nrow = T, ncol = N)
    sdt <- matrix(1, nrow = T, ncol = N)
    x22 <- x2
  } else if (DEMEAN == 1){
    # Each series is demeaned only
```

```
    mut <- matrix(rep(mean(x2), each = T), nrow = T, ncol = N)
    sdt <- matrix(1, nrow = T, ncol = N)
    x22 <- x2 - mut
  } else if (DEMEAN == 2){
    # Each series is demeaned and standardized
    mut <- matrix(rep(colMeans(x2), T), nrow = T, ncol = N, byrow = TRUE)
    sdt <- matrix(rep(sapply(x2,sd), T), nrow = T, ncol = N, byrow = TRUE)          # sd divides by T-1
    x22 <- (x2 - mut) / sdt
  } else if (DEMEAN == 3){
    # Each series is recursively demeaned and then standardized
    mut <- matrix(NA, nrow = T, ncol = N)
    for (t in 1:T) {
      mut[t, ] <- colMeans(x2[1:t, , drop = FALSE])
    }
    sdt <- matrix(rep(sd(x2),T), nrow = T, ncol = N)
    x22 <- (x2 - mut) / sdt
  }

  # Return the results
  return(list(x22 = x22, mut = mut, sdt = sdt))
}
```