Assignment_1_rxh

Rongxin Hu 2219247

2025-03-06

目录

1 Load Necessary Packages

```
library(httr)
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyr)
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

2 The FRED-MD Dataset & The Transformations

2.1 Load FRED-MD Dataset

```
url <- "https://www.stlouisfed.org/-/media/project/frbstl/stlouisfed/research/fred-md/m

response <- GET(url)
if (http_error(response)) {
  stop("Failed to download file.")
} else {
  df <- read_csv(content(response, as = "text", encoding = "UTF-8"))
}
```

```
## Rows: 794 Columns: 127
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): sasdate
## dbl (126): RPI, W875RX1, DPCERA3M086SBEA, CMRMTSPLx, RETAILx, INDPRO, IPFPNS...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
print(df)
```

```
## # A tibble: 794 x 127
##    sasdate      RPI W875RX1 DPCERA3M086SBEA CMRMTSPLx RETAILx INDPRO IPFPNSS
##    <chr>      <dbl>   <dbl>           <dbl>     <dbl>   <dbl>  <dbl>   <dbl>
##  1 Transform:     5       5               5         5       5      5       5
```

```
##  2 1/1/1959   2584.   2426            15.2   276677.  18236.   22.0    23.4
##  3 2/1/1959   2594.   2435.           15.3   278714.  18370.   22.4    23.7
##  4 3/1/1959   2610.   2453.           15.5   277775.  18523.   22.7    23.8
##  5 4/1/1959   2627.   2470            15.4   283363.  18534.   23.2    24.2
##  6 5/1/1959   2643.   2486.           15.6   285307.  18680.   23.5    24.4
##  7 6/1/1959   2651.   2494.           15.7   285280.  18850.   23.6    24.6
##  8 7/1/1959   2649.   2492            15.6   288768.  18844.   23.0    24.6
##  9 8/1/1959   2634.   2478.           15.7   273993.  18964.   22.2    24.4
## 10 9/1/1959   2636.   2478.           15.9   278039.  18716.   22.2    24.3
## # i 784 more rows
## # i 119 more variables: IPFINAL <dbl>, IPCONGD <dbl>, IPDCONGD <dbl>,
## #   IPNCONGD <dbl>, IPBUSEQ <dbl>, IPMAT <dbl>, IPDMAT <dbl>, IPNMAT <dbl>,
## #   IPMANSICS <dbl>, IPB51222S <dbl>, IPFUELS <dbl>, CUMFNS <dbl>, HWI <dbl>,
## #   HWIURATIO <dbl>, CLF16OV <dbl>, CE16OV <dbl>, UNRATE <dbl>, UEMPMEAN <dbl>,
## #   UEMPLT5 <dbl>, UEMP5TO14 <dbl>, UEMP15OV <dbl>, UEMP15T26 <dbl>,
## #   UEMP27OV <dbl>, CLAIMSx <dbl>, PAYEMS <dbl>, USGOOD <dbl>, ...
```

```
#This method bypasses the HTTP connection issues that come with R
```

2.2 Extract Transformation Codes

```
transformation_codes <- data.frame(Series = names(df)[-1], Transformation_Code = as.num
```

2.3 Function to Apply Transformations

```
mdiff <- function(x) {
    x - dplyr::lag(x, 1, default = NA)
}


apply_transformation <- function(series, code) {
  if (code == 1) {
```

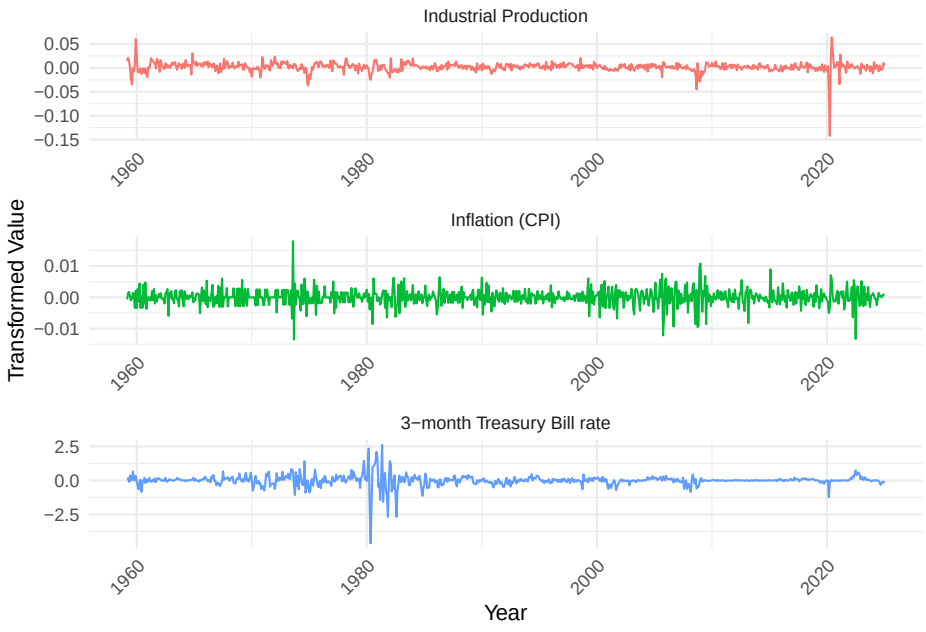
```
    return(series)
  } else if (code == 2) {
    return(mdiff(series))
  } else if (code == 3) {
    return(mdiff(mdiff(series)))
  } else if (code == 4) {
    return(log(series))
  } else if (code == 5) {
    return(mdiff(log(series)))
  } else if (code == 6) {
    return(mdiff(mdiff(log(series))))
  } else if (code == 7) {
    return(mdiff(series) / dplyr::lag(series, 1) - 1)
  } else {
    stop("Invalid transformation code")
  }
}
```

2.4 Applying the Transformations to Each Column

```
for (i in 1:nrow(transformation_codes)) {
  series_name <- transformation_codes$Series[i]
  code <- transformation_codes$Transformation_Code[i]
  df[[series_name]] <- apply_transformation(as.numeric(df[[series_name]]), code)
}

df_cleaned <- df[-c(1:3), ]
```

2.5 Plot Transformed Series

```r
series_to_plot <- c('INDPRO', 'CPIAUCSL', 'TB3MS')
series_names <- c('Industrial Production', 'Inflation (CPI)', '3-month Treasury Bill ra

plot_data <- df_cleaned %>%
  select(sasdate, all_of(series_to_plot)) %>%
  pivot_longer(-sasdate, names_to = "series", values_to = "value") %>%
  mutate(sasdate = mdy(sasdate),
         series_name = factor(series, levels = series_to_plot, labels = series_names))

ggplot(plot_data, aes(x = sasdate, y = value, color = series_name)) +
  geom_line() +
  facet_wrap(~series_name, scales = "free", ncol=1) +
  theme_minimal() +
  labs(x = "Year", y = "Transformed Value") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")
```



3 Forecasting in Time Series

3.1 Prepare Data for Estimation

```
Yraw <- df_cleaned$INDPRO
Xraw <- df_cleaned %>% select(CPIAUCSL, TB3MS)

num_lags <- 4  # this is p
num_leads <- 1  # this is h

X <- data.frame(Ones = rep(1, nrow(df_cleaned)))

for (lag in 0:num_lags) {
  X[paste0('INDPRO_lag', lag)] <- dplyr::lag(Yraw, lag)
}

for (col in names(Xraw)) {
  for (lag in 0:num_lags) {
    X[paste0(col, '_lag', lag)] <- dplyr::lag(Xraw[[col]], lag)
  }
}

y <- dplyr::lead(Yraw, num_leads)
```

3.2 Getting the Last Row fro Forecasting

```
X_T <- as.matrix(tail(X, 1))
```

3.3 Removing NA Rows (Due to Lagging/Leading)

```
complete_cases <- complete.cases(X, y)
X <- X[complete_cases, ]
y <- y[complete_cases]
```

3.4 Estimation and Forecast

```
y <- as.vector(y)
X <- as.matrix(X)

beta_ols <- solve(crossprod(X), crossprod(X, y))
```

3.5 Produce the One Step Ahead Forecast

```
forecast <- (X_T %*% beta_ols) * 100
```

4 My Forecasting Exercise

4.1 Defining Forecasting Functions (from hint)

```
calculate_msfe <- function(y, X, num_lags, h) {
  X <- as.matrix(X)
  n <- length(y)
  errors <- numeric(n - num_lags - h)

  for (t in (num_lags + 1):(n - h)) {
    # construct lag matrix
    y_lags <- embed(y[1:t], num_lags + 1)[, -1, drop = FALSE]
```

```
    X_lags <- lapply(1:ncol(X), function(i) {
      x_col <- X[1:t, i]
      embed(x_col, num_lags + 1)[, -1, drop = FALSE]
    })

    # combined lagged variables
    X_lags <- do.call(cbind, X_lags)

    #  # design mayrix (including intercept term)
    design_matrix <- cbind(1, y_lags, X_lags)

    # fix the index range of y_target
    y_target <- y[(num_lags + 1 + h):(t + h)]

    complete_cases <- complete.cases(design_matrix, y_target)
    design_matrix <- design_matrix[complete_cases, , drop = FALSE]
    y_target <- y_target[complete_cases]

    # ensure sufficient number of rows in the design matrix
    if (nrow(design_matrix) > ncol(design_matrix)) {
      beta <- solve(t(design_matrix) %*% design_matrix) %*% t(design_matrix) %*% y_targ
      X_T <- c(1, tail(y_lags, 1), tail(X_lags, 1))
      forecast <- X_T %*% beta
      errors[t - num_lags] <- (y[t + h] - forecast)^2
    } else {
      errors[t - num_lags] <- NA
    }
  }

  MSFE <- mean(errors, na.rm = TRUE)
  return(MSFE)
}
```

4.2 Different h to Our Model

```
msfe_h1 <- calculate_msfe(Yraw, Xraw, num_lags = 4, h = 1)
msfe_h4 <- calculate_msfe(Yraw, Xraw, num_lags = 4, h = 4)
msfe_h8 <- calculate_msfe(Yraw, Xraw, num_lags = 4, h = 8)

print(msfe_h1)
```

```
## [1] 8.683448e-05
```

```
print(msfe_h4)
```

```
## [1] 8.859386e-05
```

```
print(msfe_h8)
```

```
## [1] 8.449345e-05
```

4.3 Different p to Our Model

```
msfe_p1 <- calculate_msfe(Yraw, Xraw, num_lags = 1, h = 4)
msfe_p4 <- calculate_msfe(Yraw, Xraw, num_lags = 4, h = 4)
msfe_p8 <- calculate_msfe(Yraw, Xraw, num_lags = 8, h = 4)

print(msfe_p1)
```

```
## [1] 9.286144e-05
```

```
print(msfe_p4)
```

```
## [1] 8.859386e-05
```

```
print(msfe_p8)
```

```
## [1] 7.966253e-05
```