Macroeconomic Theory

Assignment I

Instructor: Prof. Eugenio Rojas

Student: C.H. (Chenhui Lu)

UFID: 76982846

Assignment 1

Instructor: Prof. Eugenio Rojas

Student: C.H.(Chenhui Lu)

UFID: 76982846

===

```
rm(list = ls())
library("data.table")
```

```
## Warning: package 'data.table' was built under R version 4.3.3
```

```
library("stargazer")
```

```
##
## Please cite as:
```

```
##  Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Tables.
```

```
##  R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

```
library("ggplot2")
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
library("tseries")
```

```
## Warning: package 'tseries' was built under R version 4.3.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library("forecast")
```

```
## Warning: package 'forecast' was built under R version 4.3.3
```

```
library("vars")
```

```
## Warning: package 'vars' was built under R version 4.3.2
```

```
## Loading required package: MASS
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:data.table':
##
##     yearmon, yearqtr
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Loading required package: urca
```

```
## Loading required package: lmtest
```

```
library("dplyr")
```

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

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

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

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

```
library("mFilter")  #  Hodrick-Prescott Filter
```

==

1 Problem 1 (50 points)

==

Use the dataset PS1.csv for this question. You may assume that the frequency is annual

```
data <- read.csv("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/PS1.csv")
data <- na.omit(data)
str(data)
```

```
## 'data.frame':    200 obs. of  2 variables:
##  $ t: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ y: num  2.09 2.14 2.04 2.09 2.05 ...
```

```
head(data)
```

```
##   t        y
## 1 1 2.093179
## 2 2 2.143852
## 3 3 2.044182
## 4 4 2.087527
## 5 5 2.049045
## 6 6 2.195577
```

```
#switch the data to be time series format
y <- ts(data$y)
```

(i) (10 points) Calculate the first 10 autocorrelations and plot the coefficients you obtain. Is the process weakly stationary?

Normally, we calculate the autocorrelations following the quitions:

```
knitr::include_graphics("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/Pics/1.1.jpe
g")
```

$$\rho(k) = corr(Y_t, Y_{t-k}) = \frac{cov(Y_t, Y_{t-k})}{\sqrt{var[Y_t]\, var[Y_{t-k}]}} = \frac{cov(Y_t, Y_{t-k})}{\sqrt{\gamma(0)\cdot\gamma(0)}} = \frac{\gamma(k)}{\gamma(0)} \quad , \text{where } \gamma(0) = var(Y_t)$$

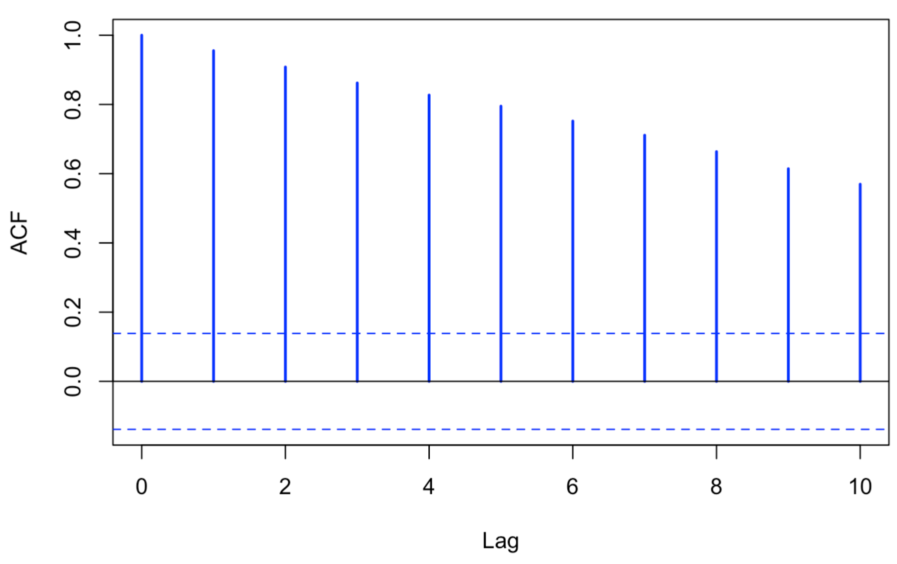
Howeverm, R has a convinient package can simply calculate the ACF.

```
print("Table 1: The First 10 Autocorrelations:")
```

```
## [1] "Table 1: The First 10 Autocorrelations:"
```

```
acf_values <- acf(y, lag.max = 10, main = "Figure 1 ACF of y (10 Lags)", col = "blue", lwd = 2)  #calculate the a
utocorelation function (ACF)
```

Figure 1 ACF of y (10 Lags)



```
print(acf_values)
```

```
##
## Autocorrelations of series 'y', by lag
##
##     0     1     2     3     4     5     6     7     8     9    10
## 1.000 0.955 0.908 0.862 0.827 0.795 0.752 0.711 0.664 0.614 0.570
```

Now we have the Table 1: The First 10 Autocorrelations and Figure 1: ACF of y (10 Lags).

Let follow the definition of Weakly Stationary to check:

1. Its unconditional expectation is constant: E[Yt] = μ for all t.
2. Its auto-covariance matrix is: γ(t, τ) = cov(Yt,Yt−τ) = E[(Yt − μ)(Yt−τ − μ)] = γ(τ) for all t, τ

3. γ(0) (= var[Yt]) is finite and not exploding.

Now, we split the dataset to be first half and second half. Sequencially, calculate the mean, variance and coefficient of autocovariance to check if they vary with time.

```
# find out the median and split the dataset to be 2 sets.
n <- length(y)
first_half <- y[1:(n/2)]
second_half <- y[(n/2 + 1):n]

# calculate the mean, variance and autocovariance
mean_full <- mean(y)
var_full <- var(y)
acf_full <- acf(y, plot = FALSE)$acf[2]  # take coefficient of lag = 1

mean_first <- mean(first_half)
var_first <- var(first_half)
acf_first <- acf(first_half, plot = FALSE)$acf[2]

mean_second <- mean(second_half)
var_second <- var(second_half)
acf_second <- acf(second_half, plot = FALSE)$acf[2]

# output
results <- data.frame(
  Statistic = c("Mean", "Variance", "ACF(1)"),
  Full = c(mean_full, var_full, acf_full),
  First_Half = c(mean_first, var_first, acf_first),
  Second_Half = c(mean_second, var_second, acf_second)
)

print(results)
```

```
##   Statistic      Full First_Half Second_Half
## 1      Mean 1.7266566  1.6963015  1.75701165
## 2  Variance 0.1243178  0.1483120  0.09971775
## 3    ACF(1) 0.9553639  0.9621588  0.94506168
```

Form the ouput, we can tell that the mean(E[Y]), variance(Yt), and ACF1(γ(1)) are far different between first half and second half. This means the time series is not weakly stationary.

In addition, we can also use ADF Test(Augmented Dickey-Fuller Test) to test if it exists the unit root to test if it is weakly stationary:

```
# ADF Test( Augmented Dickey-Fuller Test)
adf_test <- adf.test(y)
print(adf_test)
```

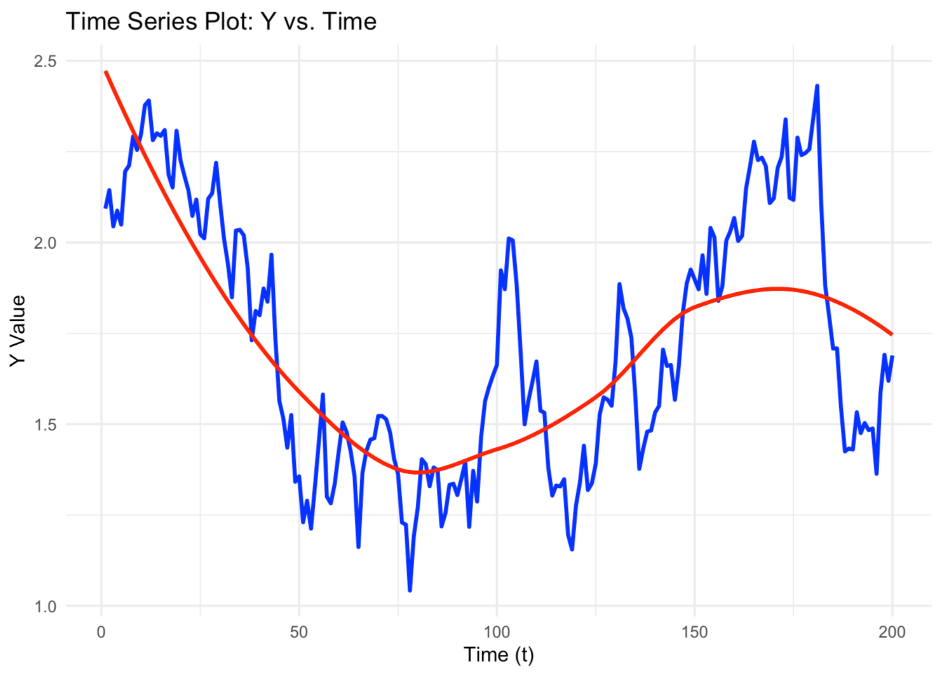
```
## 
##  Augmented Dickey-Fuller Test
## 
## data:  y
## Dickey-Fuller = -2.2654, Lag order = 5, p-value = 0.4649
## alternative hypothesis: stationary
```

Since the P-Value > 5% or even 10%, we cannot reject the hypothesis (it has the unit root), so its not weakly stationary. This test boost our analysis above.

Let's check from the time series graph and trible-check with intuition:

```
data$Time <- 1:nrow(data)

ggplot(data, aes(x = Time, y = y)) +
  geom_line(color = "blue", linewidth = 1) +
  geom_smooth(method = "loess", color = "red", se = FALSE) +
  labs(title = "Time Series Plot: Y vs. Time",
       x = "Time (t)",
       y = "Y Value") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



We can tell easily its not weakly stationary.

(ii) (10 points) Based on your results, estimate an appropriate AR(p) model via OLS and Maximum Likelihood. You may assume that the error component is a Gaussian white noise with mean 0 and variance σ^2 (Remember to also estimate σ^2.). Do your estimation results vary significantly?

we follow 3 steps:

Step 1: Use AIC to select the optimal lag order p, ensuring that the AR(p) model is neither overfitted nor underfitted.

Step 2: Once the optimal p is determined, estimate the AR(p) model using both OLS and MLE to obtain the parameters (autoregressive coefficients ϕ1, ϕ2, …ϕn, and the variance of the white noise σ^2)

Step 3: Finally, compare the results from OLS and MLE to determine whether they are consistent.

Step 1: Get best p

```
# calculate the AIC with OLS
aic_values <- sapply(1:10, function(p) {
    model <- ar.ols(y, order.max = p, demean = TRUE, intercept = TRUE)  # OLS estimate
    sigma2 <- var(model$resid, na.rm = TRUE) # variance
    n <- length(y)
    return(n * log(sigma2) + 2 * p)  # AIC calculation
})

# find the p corresponding to minimum AIC
best_p <- which.min(aic_values)
cat("optimul lag order p =", best_p, "\n")
```

```
## optimul lag order p = 1
```

Step 2: OLS Method

```
# estimation of OLS AR(best_p)
ols_model <- ar.ols(y, order.max = best_p, demean = TRUE, intercept = TRUE)

# get parameters from ols method
phi_ols <- ols_model$ar[1:best_p]  # only need lag order 1
sigma2_ols <- var(ols_model$resid, na.rm = TRUE)  # OLS variance

# Compute the final c_ols
c_ols <- mean(y) * (1 - phi_ols)

# Print the best estimates
cat(sprintf("======= AR(1) OLS =======\nc_ols = %f\nphi_mle (ϕ) = %f\nsigma^2_mle (σ^2) = %f\n",
            c_ols, phi_ols, sigma2_ols))
```

```
## ======= AR(1) OLS =======
## c_ols = 0.076970
## phi_mle (ϕ) = 0.955423
## sigma^2_mle (σ^2) = 0.010216
```

MLE Method

```r
# Corrected MLE objective function (negative log-likelihood)
mle_function <- function(y, phi, sigma2) {
  T <- length(y)

  # Compute c_mle
  y_bar <- mean(y)
  c_mle <- y_bar * (1 - phi)

  # Ensure sigma2 > 0 and 1 - phi^2 > 0
  if (sigma2 <= 0 || (1 - phi^2) <= 0) {
    return(Inf)   # Avoid log(negative) or zero, which may cause NA
  }

  # term1: Negative log-likelihood for the first observation
  term1 <- 0.5 * log(2 * pi) + 0.5 * log(sigma2 / (1 - phi^2)) +
           0.5 * ((y[1] - c_mle / (1 - phi))^2 / (sigma2 / (1 - phi^2)))

  # term2: Negative log-likelihood for the remaining observations
  residual_sum <- sum((y[2:T] - c_mle - phi * y[1:(T-1)])^2)
  term2 <- 0.5 * (T - 1) * log(2 * pi) + 0.5 * (T - 1) * log(sigma2) +
           0.5 * (residual_sum / sigma2)

  # If the computed value is NA, return Inf to avoid affecting the search
  total_nll <- term1 + term2
  if (is.na(total_nll)) {
    return(Inf)
  }

  return(total_nll)
}

# Redefine the grid search range
phi_grid <- seq(0.956037, 0.956039, by = 0.0000001)  # Refined search, to save the time, i set the range manually
sigma2_grid <- seq(0.010175, 0.010177, by = 0.000001)

# Initialize best parameters
nll_mle <- Inf  # Minimum negative log-likelihood
phi_mle <- NA
sigma2_mle <- NA

# Grid search (nested loops)
for (phi in phi_grid) {
  for (sigma2 in sigma2_grid) {
    val <- mle_function(y, phi, sigma2)

    if (!is.na(val) && val < nll_mle) {  # Ensure val is not NA
      nll_mle <- val
      phi_mle <- phi
      sigma2_mle <- sigma2
    }
  }
}

# Compute the final c_mle
best_c <- mean(y) * (1 - phi_mle)

# Print the best estimates
cat(sprintf("===== AR(1) MLE (Grid Search) =====\nc_mle = %f\nphi_mle (ϕ) = %f\nsigma^2_mle (σ^2) = %f\n",
            best_c, phi_mle, sigma2_mle))
```

```
## ===== AR(1) MLE (Grid Search) =====
## c_mle = 0.075907
## phi_mle (ϕ) = 0.956038
## sigma^2_mle (σ^2) = 0.010176
```

Keep the residual_mle for question iii) below.

```
# (2) use best_c, phi_mle, sigma2_mle calculate the fitted value and residual manually
n <- length(y)
fitted_mle <- numeric(n)

# initialize t=1
fitted_mle[1] <- best_c / (1 - phi_mle)

# from t=2 calculate y_hat[t] = c + phi*y[t-1]
for(t in 2:n){
  fitted_mle[t] <- best_c + phi_mle * y[t-1]
}

# residual: eps_t = y_t - fitted_mle[t]
residuals_mle <- y - fitted_mle
```

Step 3: Compare

```
cat(sprintf(
  "======= AR(1) OLS =======\nc_ols = %f\nphi_mle (ϕ) = %f\nsigma^2_mle (σ^2) = %f\n\n
   ===== AR(1) MLE (Grid Search) =====\nc_mle = %f\nphi_mle (ϕ) = %f\nsigma^2_mle (σ^2) = %f\n",
  c_ols, phi_ols, sigma2_ols, best_c, phi_mle, sigma2_mle
))
```

```
## ======= AR(1) OLS =======
## c_ols = 0.076970
## phi_mle (ϕ) = 0.955423
## sigma^2_mle (σ^2) = 0.010216
##
##
##    ===== AR(1) MLE (Grid Search) =====
## c_mle = 0.075907
## phi_mle (ϕ) = 0.956038
## sigma^2_mle (σ^2) = 0.010176
```

We can tell, the estimation results are close but not exactly the same.

(iii) (10 points) Choose your preferred estimation method. Are the residuals of the estimated model white noise? Be specific.

I decide to use MLE.

```
knitr::include_graphics("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/Pics/1.3.png
")
```

$$\hat{\varepsilon}_t = y_t - \hat{\phi} y_{t-1}$$

For εt to be white noise, it need to satisfy 3 condition: 1. E[εt] = 0, 2. E[ε2t] = σ2 < ∞, 3. cov(εt, εt−k) = 0, ∀ k/= 0

Step 1, get the statistics of residual of AR(1)

```
# we have got residuals_mle from ii)

# MLE residual's mean and variance
cat(sprintf("MLE residual mean: %f\nMLE residual variance: %f\n",
            mean(residuals_mle), var(residuals_mle)))
```

```
## MLE residual mean: -0.000181
## MLE residual variance: 0.010844
```

Since the E[εt] = -0.0001812651 close to 0 and E[ε2t] = 0.01084384, close to σ^2 = 0.01035527, the condition 1 and 2 have been satisfied.

Step 2:

```
# calculate the covariance metrix with different lag k
max_lag <- 10  # calculate from 1 to 10 order lag k
cov_matrix_mle <- sapply(1:max_lag, function(k) {
  return(cov(residuals_mle[1:(length(residuals_mle)-k)], residuals_mle[(k+1):length(residuals_mle)]))
})

# Output covariance matrix from lag 1 to 10
cat("===== Covariance Matrix of MLE Residuals from Lag 1 to 10 =====\n")
```

```
## ===== Covariance Matrix of MLE Residuals from Lag 1 to 10 =====
```

```
names(cov_matrix_mle) <- paste0("Lag_", 1:max_lag)
print(cov_matrix_mle)
```

```
##         Lag_1         Lag_2         Lag_3         Lag_4         Lag_5
##  5.253663e-04  9.383318e-05 -1.143584e-03 -1.928436e-04  1.552156e-03
##         Lag_6         Lag_7         Lag_8         Lag_9        Lag_10
## -7.760236e-05  9.967168e-04  3.841960e-04 -4.102972e-04 -1.112489e-03
```

They are all very small, which mean the condition 3 also has been satisfied. Overall, all 3 conditions have been satisfied, the error term has been proved to be white noise.

In addition, we can take Ljung-Box test to double- check our result. H0::all autocorrelation coefficients ρk = 0, ∀k

```
 Box.test(residuals_mle, lag = 10, type = "Ljung-Box")
```

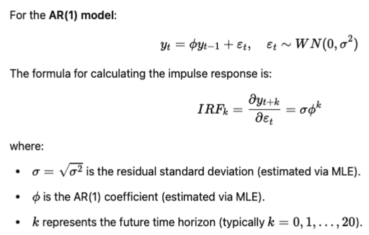
```
##
##  Box-Ljung test
##
## data:  residuals_mle
## X-squared = 10.99, df = 10, p-value = 0.3583
```

Since the p-value > 0.05 or 0.1, we cannot reject the null. The error term is white noise.

(iv) (10 points) Compute the impulse response function of your model to a one standard deviation shock.

The impulse response function (IRF) measures the impact of a one-standard-deviation shock occurring at time t on future periods k.

```
knitr::include_graphics("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/Pics/1.4.png
")
```



For the **AR(1) model**:

$$y_t = \phi y_{t-1} + \varepsilon_t, \quad \varepsilon_t \sim WN(0, \sigma^2)$$

The formula for calculating the impulse response is:

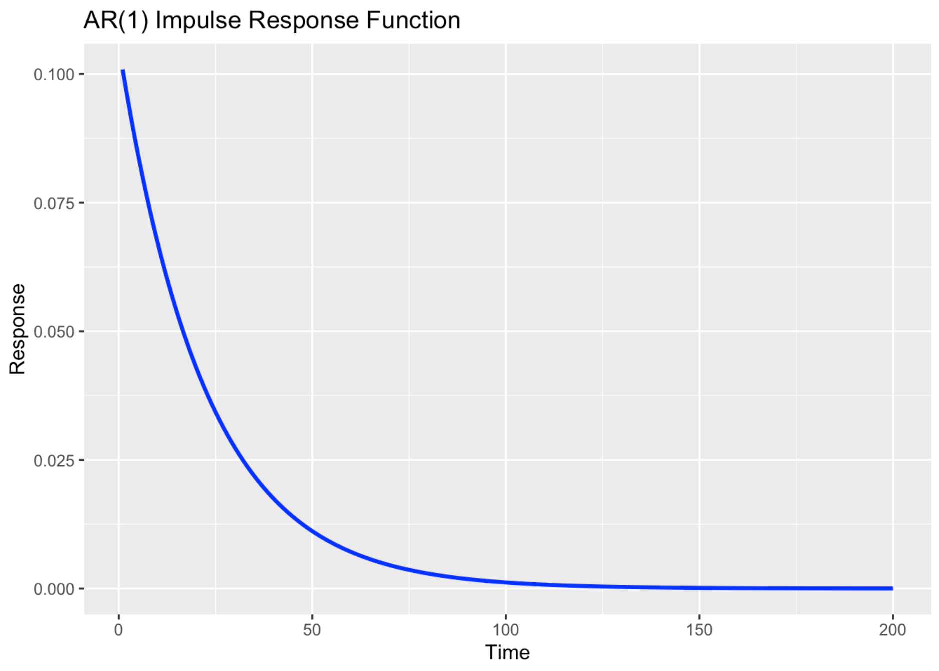
$$IRF_k = \frac{\partial y_{t+k}}{\partial \varepsilon_t} = \sigma \phi^k$$

where:

- $\sigma = \sqrt{\sigma^2}$ is the residual standard deviation (estimated via MLE).
- ϕ is the AR(1) coefficient (estimated via MLE).
- k represents the future time horizon (typically $k = 0, 1, \ldots, 20$).

We already have the phi_mle (ϕ) = 0.9560379, sigma^2_mle (σ^2) = 0.010176 from question (ii) step2.

```
# Compute standard deviation shock
shock <- sqrt(sigma2_mle)
impulse_response <- numeric(length(y))
impulse_response[1] <- shock

# Compute the impulse response function for an AR(1) process
for (i in 2:length(y)) {
  impulse_response[i] <- phi_mle * impulse_response[i-1]
}

# Compute impulse response function
ggplot(data.frame(time = 1:length(y), response = impulse_response), aes(x = time, y = response)) +
  geom_line(color = "blue", linewidth = 1) +
  ggtitle("AR(1) Impulse Response Function") +
  xlab("Time") +
  ylab("Response")
```



From the figure, after a one-standard-deviation positive shock, the series initially jumps to about 0.10 and then decays exponentially back toward zero. We can know for this AR(1) process: the shock has a clear short-term impact but eventually dies out, bringing the series back to its mean.

(v) (10 points) Re-estimate but now leaving the last 10 periods out of your sample. Using your model, forecast the evolution of the dependent variable for the next 10 periods, and compare it with the actual sample. Is your model doing a good job in terms of explaining the dynamics of the time series?

Step 1. Remove the last 10 observations

```
# Remove the last 10 observations
train_length <- length(y) - 10
train_data <- y[1:train_length]  # Training set
test_data <- y[(train_length + 1):length(y)]  # Test set (actual values)

# Print data splitting results
cat(sprintf("Training set length: %d\nTest set length: %d\n", length(train_data), length(test_data)))
```

```
## Training set length: 190
## Test set length: 10
```

Step 2. MLE estimation of parameters：

```r
# MLE estimation of phi and sigma^2
best_nll_new <- Inf
best_phi_new <- 0.02
best_sigma2_new <- 0.01

# Choose the grid search range
phi_grid <- seq(0.95, 0.96, by = 0.0001)  # estimation range
sigma2_grid <- seq(0.01, 0.02, by = 0.0001)

# MLE objective function (negative log-likelihood)
mle_function <- function(y, phi, sigma2) {
  T <- length(y)
  y_bar <- mean(y)
  c_mle <- y_bar * (1 - phi)

  # Ensure sigma2 > 0 and (1 - phi^2) > 0
  if (sigma2 <= 0 || (1 - phi^2) <= 0) {
    return(Inf)
  }

  term1 <- 0.5 * log(2 * pi) + 0.5 * log(sigma2 / (1 - phi^2)) +
           0.5 * ((y[1] - c_mle / (1 - phi))^2 / (sigma2 / (1 - phi^2)))

  residual_sum <- sum((y[2:T] - c_mle - phi * y[1:(T-1)])^2)
  term2 <- 0.5 * (T - 1) * log(2 * pi) + 0.5 * (T - 1) * log(sigma2) +
           0.5 * (residual_sum / sigma2)

  total_nll <- term1 + term2
  return(ifelse(is.na(total_nll), Inf, total_nll))
}

# dp MLE estimation
for (phi in phi_grid) {
  for (s2 in sigma2_grid) {
    val <- mle_function(train_data, phi, s2)
    if (!is.na(val) && val < best_nll_new) {
      best_nll_new <- val
      best_phi_new <- phi
      best_sigma2_new <- s2
    }
  }
}

# Print the estimation results
cat(sprintf("===== Re-estimated AR(1) MLE (Grid Search) =====\n\nphi_tilde = %f\nsigma^2_tilde = %f\n",
            best_phi_new, best_sigma2_new))
```

```
## ===== Re-estimated AR(1) MLE (Grid Search) =====
##
## phi_tilde = 0.958600
## sigma^2_tilde = 0.010200
```

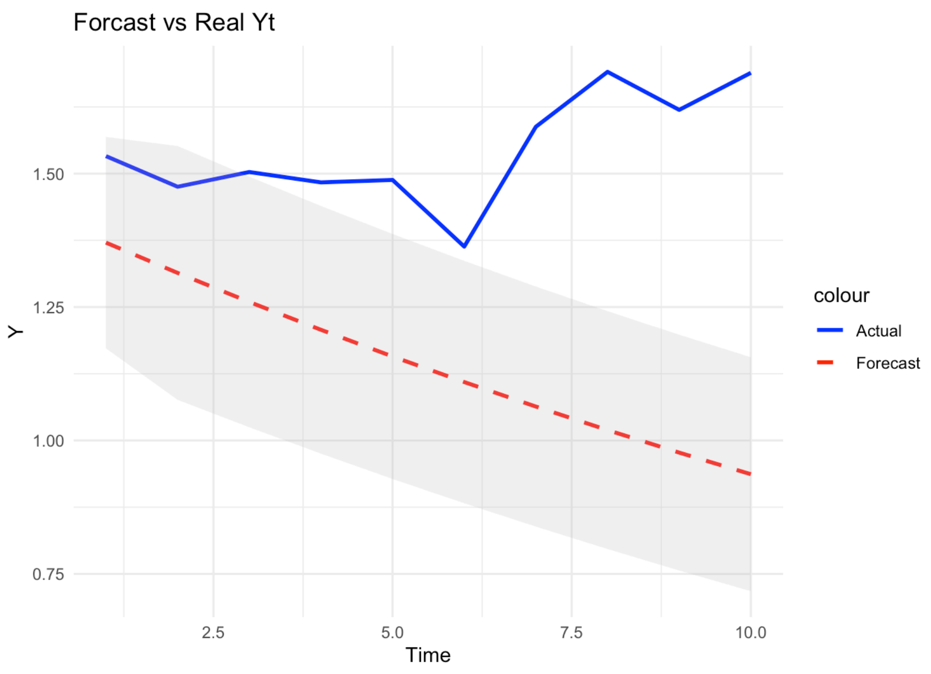
Step 3. Forecast

```
# Initialize an array for forecast values
y_forecast <- numeric(10)

# Forecast Yt
y_forecast[1] <- best_phi_new * train_data[length(train_data)]
for (t in 2:10) {
  y_forecast[t] <- best_phi_new * y_forecast[t-1]
}

# Actual values
y_actual <- y[(length(y)-9):length(y)]

# Calculate forecast errors
forecast_error <- y_actual - y_forecast

# Calculate 95% confidence intervals
upper_bound <- numeric(10)
lower_bound <- numeric(10)

upper_bound[1] <- best_phi_new * train_data[length(train_data)] + 1.96 * sqrt(best_sigma2_new)
lower_bound[1] <- best_phi_new * train_data[length(train_data)] - 1.96 * sqrt(best_sigma2_new)

for (k in 2:10) {
  upper_bound[k] <- best_phi_new^k * train_data[length(train_data)] +
    1.96 * sqrt((best_sigma2_new * (1 + best_phi_new^(2*k) / (1 + best_phi_new^2))))
  lower_bound[k] <- best_phi_new^k * train_data[length(train_data)] -
    1.96 * sqrt((best_sigma2_new * (1 + best_phi_new^(2*k) / (1 + best_phi_new^2))))
}

# Print forecast results
forecast_results <- data.frame(
  t = 1:10,
  y_forecast = y_forecast,
  y_actual = y_actual,
  forecast_error = forecast_error,
  upper_bound = upper_bound,
  lower_bound = lower_bound
)
print(forecast_results)
```

```
##     t y_forecast y_actual forecast_error upper_bound lower_bound
## 1   1  1.3706754 1.532697      0.1620216    1.568626   1.1727251
## 2   2  1.3139294 1.475356      0.1614261    1.551473   1.0763856
## 3   3  1.2595327 1.502921      0.2433885    1.494115   1.0249503
## 4   4  1.2073881 1.483501      0.2761133    1.439216   0.9755602
## 5   5  1.1574022 1.488066      0.3306637    1.386670   0.9281348
## 6   6  1.1094858 1.363232      0.2537464    1.336375   0.8825966
## 7   7  1.0635531 1.587733      0.5241803    1.288235   0.8388716
## 8   8  1.0195220 1.690563      0.6710411    1.242156   0.7968884
## 9   9  0.9773138 1.619316      0.6420019    1.198049   0.7565788
## 10 10  0.9368530 1.688737      0.7518835    1.155829   0.7178772
```

Step 4. Visualize the forecast results

```
# Forcast vs. Real Yt
ggplot(forecast_results, aes(x = t)) +
  geom_line(aes(y = y_actual, color = "Actual"), linewidth = 1) +
  geom_line(aes(y = y_forecast, color = "Forecast"), linewidth = 1, linetype = "dashed") +
  geom_ribbon(aes(ymin = lower_bound, ymax = upper_bound), alpha = 0.2, fill = "gray") +
  labs(title = "Forcast vs Real Yt", x = "Time", y = "Y") +
  scale_color_manual(values = c("Actual" = "blue", "Forecast" = "red")) +
  theme_minimal()
```



==

2 Problem 2 (50 points)

==

```
# Empty Environment
rm(list = ls())

data <- read.csv("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/PS2.csv")
data <- na.omit(data)

# order the time
data <- data[order(data$t), ]
head(data)
```

```
##   t      RGDP     GPDI FFER
## 1 1 10047.386 1304.586 8.25
## 2 2 10083.855 1304.993 8.24
## 3 3 10090.569 1279.695 8.16
## 4 4  9998.704 1208.107 7.74
## 5 5  9951.916 1167.908 6.43
## 6 6 10029.510 1167.904 5.86
```

Data Description:

1. t: time index unit in one quarter form 1990Q1 to 2019Q4
2. RGD: US real GDP
3. GPDI: Real Gross Private Domestic Investment
4. FFER: Average Federal Funds Effective Rate (in every quarter)

(i) (10 points) Use the Hodrick-Prescott filter on the GDP and Investment time series. Extract the cyclical component of each one of them. Present summary statistics (mean, standard deviation, first order autocorrelation) for these time series. Use them for the rest of the question.

```
# Set the HP filter parameter for quarterly data
lambda <- 1600

# Apply the HP filter
hp_gdp <- hpfilter(data$RGDP, freq = lambda)
hp_gpdi <- hpfilter(data$GPDI, freq = lambda)

# Extract the cyclical components
gdp_cycle <- hp_gdp$cycle
gpdi_cycle <- hp_gpdi$cycle

# Compute descriptive statistics: mean, standard deviation, and first-order autocorrelation
summary_stats <- data.frame(
  Variable = c("GDP Cycle", "Investment Cycle"),
  Mean = c(mean(gdp_cycle), mean(gpdi_cycle)),
  Std_Dev = c(sd(gdp_cycle), sd(gpdi_cycle)),
  Autocorrelation_1 = c(acf(gdp_cycle, plot = FALSE)$acf[2], acf(gpdi_cycle, plot = FALSE)$acf[2])
)

print(summary_stats)
```

```
##           Variable          Mean  Std_Dev Autocorrelation_1
## 1        GDP Cycle -6.840477e-15 155.6897         0.8614700
## 2 Investment Cycle  1.651058e-14 128.6604         0.8827282
```

Mean:

GDP Cycle: -6.84e-15, Investment Cycle: 1.65e-14

These values are very close to zero, indicating that the cyclical components extracted by the HP filter essentially have a mean of zero, which is consistent with the idea of removing the long-term trend.

Standard Deviation:

GDP Cycle: 155.6897, Investment Cycle: 128.6604

This suggests the GDP cycle has greater volatility than the investment cycle (i.e., a higher standard deviation). A larger standard deviation indicates stronger fluctuations in the time series.

First-Order Autocorrelation:

GDP Cycle: 0.8614, Investment Cycle: 0.8827

Both coefficients are close to 1, signifying a strong temporal dependence: the current period’s value is highly correlated with the previous period’s value. This aligns well with the intuitive notion of economic cycles having persistent effects over time.

(ii) (20 points) Estimate a VAR(2) that includes GDP, investment and the federal funds rate. Present and explain your estimation results.

Because question (i) only requires using the Hodrick-Prescott filter on the GDP and Investment time series, not the federal funds rate, but VAR() requires the time series to be stationary, I decided to perform unit root tests to ensure all three time series are stationary.

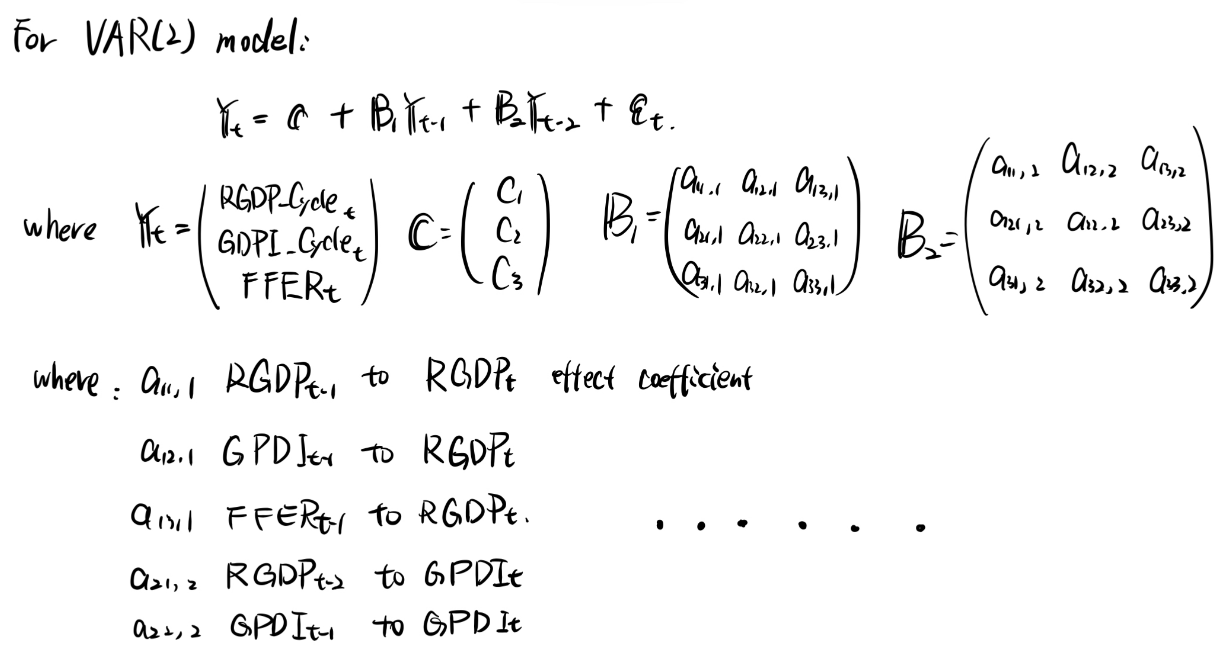
```
# Construct the data required for VAR(2) (using the previously obtained gdp_cycle and gpdi_cycle)
var_data <- data.frame(
  RGDP_Cycle = gdp_cycle,   # Cyclical component of GDP after filtering
  GPDI_Cycle = gpdi_cycle,  # Cyclical component of Investment after filtering
  FFER = data$FFER          # Federal Funds Effective Rate (keeping the original variable name)
)

# ADF Unit Root Test to ensure stationarity

adf_rGDP <- ur.df(var_data$RGDP_Cycle, type = "drift", selectlags = "AIC")
adf_gpdi <- ur.df(var_data$GPDI_Cycle, type = "drift", selectlags = "AIC")
adf_ffer <- ur.df(var_data$FFER, type = "drift", selectlags = "AIC")

# Output the ADF test results
cat("===== ADF Unit Root Test Results:===== \n",
    paste(capture.output(summary(adf_rGDP)), collapse = "\n"), "\n\n",
    paste(capture.output(summary(adf_gpdi)), collapse = "\n"), "\n\n",
    paste(capture.output(summary(adf_ffer)), collapse = "\n"), "\n")
```

```
## ===== ADF Unit Root Test Results:=====
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression drift
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + z.diff.lag)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -320.98  -44.94   -0.02   46.15  195.42
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.05852    6.69621  -0.158 0.874673
## z.lag.1     -0.17354    0.04463  -3.888 0.000169 ***
## z.diff.lag   0.31300    0.08774   3.567 0.000527 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 72.72 on 115 degrees of freedom
## Multiple R-squared:  0.1639, Adjusted R-squared:  0.1493
## F-statistic: 11.27 on 2 and 115 DF,  p-value: 3.394e-05
##
##
## Value of test-statistic is: -3.8882 7.5637
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau2 -3.46 -2.88 -2.57
## phi1  6.52  4.63  3.81
##
##
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression drift
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + z.diff.lag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -204.968  -32.370   -0.199   35.882  145.178
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.42480    5.23652  -0.272 0.786041
## z.lag.1     -0.14805    0.04185  -3.538 0.000583 ***
## z.diff.lag   0.31105    0.08895   3.497 0.000670 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 56.87 on 115 degrees of freedom
## Multiple R-squared:  0.1494, Adjusted R-squared:  0.1346
## F-statistic:  10.1 on 2 and 115 DF,  p-value: 9.098e-05
##
##
## Value of test-statistic is: -3.538 6.2954
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau2 -3.46 -2.88 -2.57
## phi1  6.52  4.63  3.81
##
##
```

```
##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression drift
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 + 1 + z.diff.lag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.33781 -0.09856 -0.02698  0.16081  0.61194
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.07525    0.04346   1.731  0.08608 .
## z.lag.1     -0.03349    0.01176  -2.848  0.00521 **
## z.diff.lag   0.68319    0.06604  10.345  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2969 on 115 degrees of freedom
## Multiple R-squared:  0.505,  Adjusted R-squared:  0.4964
## F-statistic: 58.66 on 2 and 115 DF,  p-value: < 2.2e-16
##
##
## Value of test-statistic is: -2.8479 4.3325
##
## Critical values for test statistics:
##       1pct  5pct 10pct
## tau2 -3.46 -2.88 -2.57
## phi1  6.52  4.63  3.81
##
```

Since the P-value are all very small, we can sure that three variables, especially original FFER, are all stationary, so we can use them to make VAR(2)

```
knitr::include_graphics("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/Pics/2.2.jpe
g")
```



For VAR(2) model:

$$\mathbb{Y}_t = \mathbb{C} + \mathbb{B}_1 \mathbb{Y}_{t-1} + \mathbb{B}_2 \mathbb{Y}_{t-2} + \mathcal{E}_t.$$

where $\mathbb{Y}_t = \begin{pmatrix} RGDP\_Cycle_t \\ GDPI\_Cycle_t \\ FFER_t \end{pmatrix}$ $\mathbb{C} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$ $\mathbb{B}_1 = \begin{pmatrix} a_{11,1} & a_{12,1} & a_{13,1} \\ a_{21,1} & a_{22,1} & a_{23,1} \\ a_{31,1} & a_{32,1} & a_{33,1} \end{pmatrix}$ $\mathbb{B}_2 = \begin{pmatrix} a_{11,2} & a_{12,2} & a_{13,2} \\ a_{21,2} & a_{22,2} & a_{23,2} \\ a_{31,2} & a_{32,2} & a_{33,2} \end{pmatrix}$

where: $a_{11,1}$ $RGDP_{t-1}$ to $RGDP_t$ effect coefficient

$a_{12,1}$ $GPDI_{t-1}$ to $RGDP_t$

$a_{13,1}$ $FFER_{t-1}$ to $RGDP_t$. . . . . . .

$a_{21,2}$ $RGDP_{t-2}$ to $GPDI_t$

$a_{22,2}$ $GPDI_{t-1}$ to $GPDI_t$

I will use two methods:

Method 1: Manual calculation.

Method 2: Using R’s built-in package functions.：

Method 1: Manual calculation.

```
# ==========================
#Generate VAR(2) variables
# ==========================

  # Create Y (dependent variables): current values
  RGDP_Cycle  <- var_data$RGDP_Cycle
  GPDI_Cycle  <- var_data$GPDI_Cycle
  FFER        <- var_data$FFER

  Y <- cbind(RGDP_Cycle, GPDI_Cycle, FFER)

  # Create X (independent variables): values lagged by 1 period and 2 periods
  X <- cbind(lag(RGDP_Cycle, 1), lag(GPDI_Cycle, 1), lag(FFER, 1),
            lag(RGDP_Cycle, 2), lag(GPDI_Cycle, 2), lag(FFER, 2))

# Remove NA values (the first two rows become NA due to lagging)
data_VAR <- na.omit(data.frame(Y, X))

# Redefine Y and X (to ensure there are no NA values)
Y <- as.matrix(data_VAR[, 1:3])  # RGDP_Cycle, GPDI_Cycle, FFER
X <- as.matrix(data_VAR[, 4:9])  # Lagged values (1 period and 2 periods)

# Add a constant term
X <- cbind(X, 1)

# ==========================
# Estimate VAR(2) using OLS
# ==========================
B <- solve(t(X) %*% X) %*% t(X) %*% Y  # 计算 OLS 估计值

# Assign column and row names to the matrix
colnames(B) <- c("RGDP_Cycle", "GPDI_Cycle", "FFER")
rownames(B) <- c("RGDP_Lag1", "GPDI_Lag1", "FFER_Lag1",
                 "RGDP_Lag2", "GPDI_Lag2", "FFER_Lag2",
                 "Constant")

# Print the coefficient matrix from the OLS estimates
print(B)
```

```
##            RGDP_Cycle  GPDI_Cycle          FFER
## RGDP_Lag1   0.9311275   0.1867617  0.0003625328
## GPDI_Lag1   0.1925060   0.8747483 -0.0003131217
## FFER_Lag1  18.3901812  27.3555622  1.6234940881
## RGDP_Lag2  -0.3914145  -0.3270807 -0.0012866156
## GPDI_Lag2   0.1569163   0.1021688  0.0011182725
## FFER_Lag2 -15.9812617 -25.9614647 -0.6481806605
## Constant   -7.8916498  -4.4838220  0.0462590117
```

Method 2: Using R’s package functions

```
# ==========================
# Estimate the VAR(2) model
# ==========================
var_model <- VAR(var_data, p = 2, type = "const")

# Making Table
# Extract the three model objects:
mod_rgdpc <- var_model$varresult$RGDP_Cycle
mod_gpdic <- var_model$varresult$GPDI_Cycle
mod_ffer  <- var_model$varresult$FFER

# Directly pass them to stargazer
stargazer(
  mod_rgdpc,
  mod_gpdic,
  mod_ffer,
  type = "text",
  column.labels = c("RGDP Cycle", "GPDI Cycle", "FFER"),  # 3 column labels
  dep.var.labels.include = FALSE, # Hide "Dependent variable: y"
  model.numbers = FALSE           # Hide model numbers (1) (2) (3)
)
```

```
##
## ===================================================================
##                                        Dependent variable:
##                                  ----------------------------------
##                                  RGDP Cycle GPDI Cycle     FFER
## -------------------------------------------------------------------
## RGDP_Cycle.l1                     0.931***    0.187       0.0004
##                                   (0.151)    (0.115)     (0.001)
##
## GPDI_Cycle.l1                      0.193     0.875***    -0.0003
##                                   (0.197)    (0.151)     (0.001)
##
## FFER.l1                            18.390    27.356*     1.623***
##                                   (18.231)   (13.982)    (0.075)
##
## RGDP_Cycle.l2                     -0.391**  -0.327***    -0.001**
##                                   (0.150)    (0.115)     (0.001)
##
## GPDI_Cycle.l2                      0.157      0.102       0.001
##                                   (0.196)    (0.150)     (0.001)
##
## FFER.l2                           -15.981    -25.961*   -0.648***
##                                   (17.845)   (13.686)    (0.074)
##
## const                              -7.892     -4.484      0.046
##                                   (11.148)   (8.550)     (0.046)
##
## -------------------------------------------------------------------
## Observations                        118        118         118
## R2                                 0.798      0.829       0.984
## Adjusted R2                        0.787      0.820       0.983
## Residual Std. Error (df = 111)   71.123     54.549      0.294
## F Statistic (df = 6; 111)      72.947***  89.882***  1,159.109***
## ===================================================================
## Note:                                  *p<0.1; **p<0.05; ***p<0.01
```

Two method get the same results.

```
knitr::include_graphics("/Users/terrylu/Desktop/UF/Courses/2024-2025/2025_Spring/Macro/Problem_Set_1/Pics/2.3.jpe
g")
```

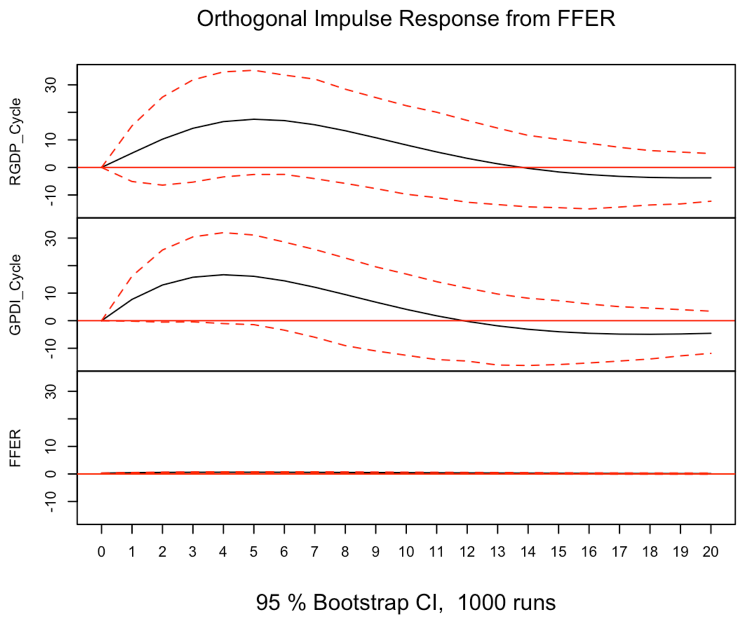
So, VAR(2) model:

$$\mathbb{Y}_t = \mathbb{C} + \mathbb{B}_1 \mathbb{Y}_{t-1} + \mathbb{B}_2 \mathbb{Y}_{t-2} + \mathcal{E}_t.$$

where: $\mathbb{C} = \begin{pmatrix} -7.892 \\ -4.484 \\ 0.046 \end{pmatrix}$ $\mathbb{B}_1 = \begin{pmatrix} 0.931 & 0.183 & 18.390 \\ 0.187 & 0.875 & 27.356 \\ 0.0004 & -0.0003 & 1.623 \end{pmatrix}$ $\mathbb{B}_2 = \begin{pmatrix} -0.391 & 0.157 & -15.981 \\ -0.327 & 0.102 & -25.961 \\ -0.001 & 0.001 & -0.648 \end{pmatrix}$

Building the model I'm working through building a VAR(2) model using GDP, Investment, and FFER(Federal Funds Effective Rate). I'll use impulse response functions to observe FFER shocks over 20 periods, incorporating cumulative effects and bootstrapping.

```
# set order with gdp_cycle, gpdi_cycle, FFER
var_data_1 <- data.frame(
  RGDP_Cycle  = gdp_cycle,      # 1. GDP
  GPDI_Cycle = gpdi_cycle,     # 2. Invest
  FFER       = data$FFER       # 3. Federal Rate
)

# Estimate VAR(2) model
var_model_1 <- VAR(var_data_1, p = 2, type = "const")

# Compute the impulse response for a "1 standard deviation shock" to FFER over 20 periods
irf_result_1 <- irf(
  var_model_1,
  impulse  = "FFER",                 # Shock to FFER
  response = c("RGDP_Cycle", "GPDI_Cycle", "FFER"),
  n.ahead  = 20,
  ortho    = TRUE,                   # Cholesky decomposition (orthogonalized shock)
  boot     = TRUE,                   # Bootstrap method
  runs     = 1000                    # Number of bootstrap replications (for confidence intervals)
)

# (4) Plot the impulse response function
plot(irf_result_1)
```



Immediate effect of the Shock:

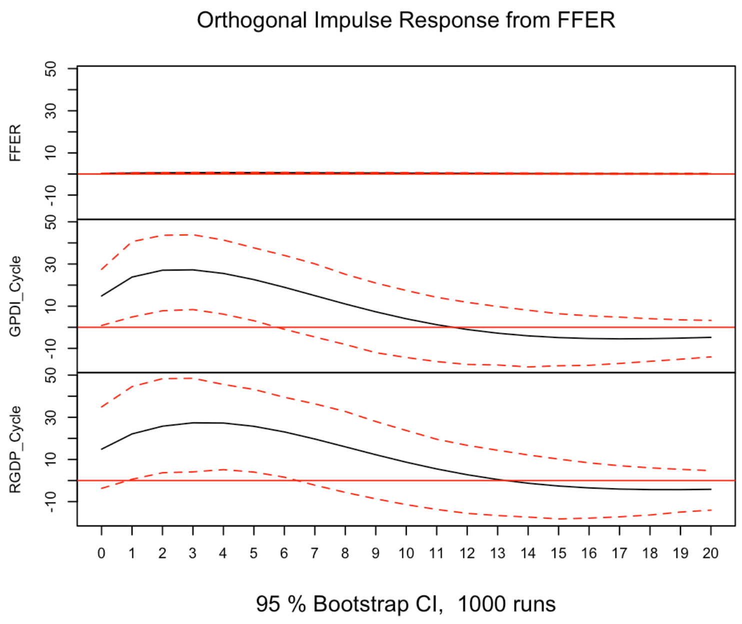
From the graph, we observe that after a positive shock to the federal funds rate (FFER), GDP and investment exhibit a strong and noticeable positive deviation in the first few periods (approximately 0 to 3 quarters). This implies that in the short term, both variables rise or remain at elevated levels. This aligns with economic intuition, as the Federal Reserve typically raises interest rates during periods of economic overheating, when GDP is growing rapidly due to inflation and the investment market is booming.

Long-Term Effects of the Shock:

As time progresses (around 5 to 10 quarters later), the response curves of GDP and investment begin to decline and gradually return to the zero axis, or even turn negative. This indicates that the monetary shock has only a temporary effect on economic activity. In the long run, both variables revert to their original trends, and after several quarters, they may even drop slightly below their initial levels. This is consistent with the real-world observation that interest rate hikes, over the long term, tend to cool down the economy, slowing GDP growth and reducing investment activity.

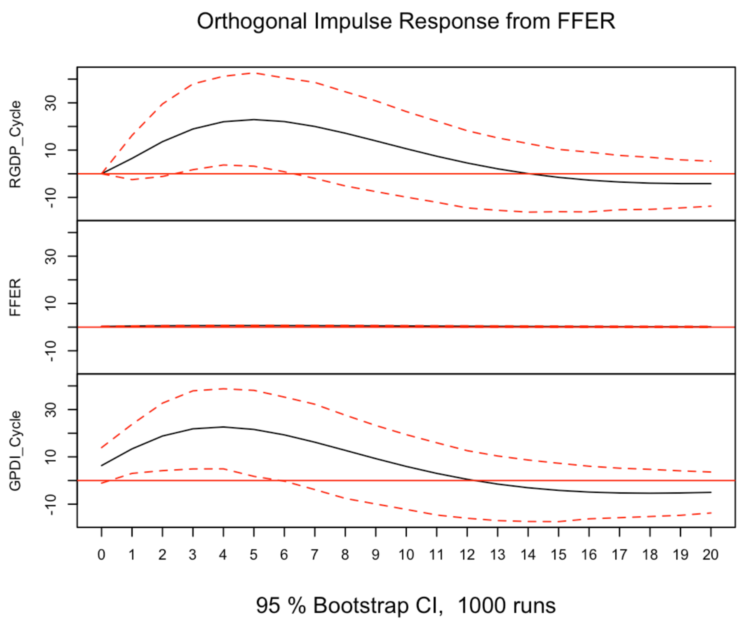
Extension 1: Now, we change the ordering of variables to FFER(Federal Rate) → Investment → GDP, assuming that monetary policy is implemented first, leading to changes in investment behavior, which ultimately affect GDP.

```
# set order with FFER, GPDI_Cycle, RGDP_Cycle
var_data_2 <- data.frame(
  FFER       = data$FFER,      # 1. FFER
  GPDI_Cycle = gpdi_cycle,    # 2. Investment
  RGDP_Cycle  = gdp_cycle     # 3. GDP
)

# (2) Estimate VAR(2) model
var_model_2 <- VAR(var_data_2, p = 2, type = "const")

# (3) Compute the impulse response for a "1 standard deviation shock" to FFER over 20 periods
irf_result_2 <- irf(
  var_model_2,
  impulse  = "FFER",                # Shock to FFER
  response = c("RGDP_Cycle", "GPDI_Cycle", "FFER"),
  n.ahead  = 20,
  ortho    = TRUE,
  boot     = TRUE,
  runs     = 1000
)

# (4) Plot
plot(irf_result_2)
```



Extension 2: Now, we change the ordering of variables to GDP → FFER → Investment, assuming that the Federal Reserve first observes GDP performance, then adjusts the Federal Funds Effective Rate policy accordingly, which subsequently influences investment behavior.

```
# set order with RGDP_Cycle, FFER, GPDI_Cycle
var_data_3 <- data.frame(
  RGDP_Cycle  = gdp_cycle,      # 1. GDP
  FFER       = data$FFER,       # 2. FFER
  GPDI_Cycle = gpdi_cycle       # 3. Investment
)

# Estimate VAR(2) model
var_model_3 <- VAR(var_data_3, p = 2, type = "const")

# Compute the impulse response for a "1 standard deviation shock" to FFER over 20 periods
irf_result_3 <- irf(
  var_model_3,
  impulse  = "FFER",
  response = c("RGDP_Cycle", "GPDI_Cycle", "FFER"),
  n.ahead  = 20,
  ortho    = TRUE,
  boot     = TRUE,
  runs     = 1000
)

# Plot
plot(irf_result_3)
```



Overall Conclusion: In the short term, the shock has an immediate impact, but over the long run, the response gradually weakens and eventually returns to its initial level, without altering the overall trend.

However, in Extension 1 and Extension 2, there is an immediate jump in response, whereas the original ordering I selected does not exhibit such a sudden spike.