FA542 - Final Exam

I pledge my honor that I have abided by the Stevens Honor System.

Sid Bhatia

2023-12-13

Problem 1 (50pt)

Suppose that the daily log return of a pair of securities follows the following model:

$$\begin{cases} r_{1,t} = -0.05 + 0.1r_{1,t-1} + 0.05r_{2,t-1} + a_{1,t} \\ r_{2,t} = 0.1 - 0.1r_{1,t-1} + 0.3r_{2,t-1} + a_{2,t} \end{cases}$$

where a_t denotes a bivariate normal distribution with mean 0 and covariance:

$$\Sigma := \begin{pmatrix} 0.4 & -0.1 \\ -0.1 & 0.2 \end{pmatrix}$$

Any matrix operations can be computed in R. If any formulas require infinite series, you may approximate using the first 5 terms.

a. Verify that the return series r_t is a weakly stationary process.

Hint: The "polyroot" function can be used to find all roots of a polynomial in R and the "eigen" function can be used to find all eigenvalues of a matrix in R.

```
# Defining the coefficient matrix for the mean calculation
A <- matrix(c(1 - 0.1, -0.05, 0.1, 1 - 0.3), nrow = 2, byrow = TRUE)

# Defining the constant terms
B <- c(-0.05, 0.1)

# Solving for mu (the mean of r1 and r2).
mu <- solve(A, B)

# Defining the coefficient matrix for the characteristic equation.
char_matrix <- matrix(c(-0.1, 0.1, -0.05, -0.3), nrow = 2, byrow = TRUE)

# Calculating the eigenvalues of the characteristic matrix.
eigenvalues <- eigen(char_matrix)$values

# Display the results.
print("Means of r1 and r2:")
```

```
## [1] "Means of r1 and r2:"
```

```
mu
```

```
## [1] -0.04724409  0.14960630
```

```r
print("Eigenvalues of the characteristic matrix:")
```

```
## [1] "Eigenvalues of the characteristic matrix:"
```

```r
eigenvalues
```

```
## [1] -0.2707107 -0.1292893
```

```r
# Check if the absolute values of eigenvalues are less than 1 (inside the unit circle).
is_stationary <- all(abs(eigenvalues) < 1)

# Display the conclusion.
if (is_stationary)
{
  print("The series is weakly stationary.")
} else
{
  print("The series is not weakly stationary.")
}
```

```
## [1] "The series is weakly stationary."
```

b.

i. What is the mean vector of the return series r_t?

```r
mu
```

```
## [1] -0.04724409  0.14960630
```

ii. What is the covariance matrix of the return series r_t?

```r
library(MASS) # For solving Yule-Walker equations

# Covariance matrix
Sigma <- matrix(c(0.4, -0.1, -0.1, 0.2), nrow = 2, byrow = TRUE)
phi <- matrix(c(0.1, 0.05, -0.1, 0.3), nrow = 2, byrow = TRUE)

# Solving Yule-Walker equations
cov_matrix <- solve(toeplitz(1:2), Sigma)

cov_matrix
```

```
##      [,1]       [,2]
## [1,] -0.2  0.1666667
## [2,]  0.3 -0.1333333
```

iii. What are the lag-1, lag-2, and lag-5 cross-correlation matrices of the return series r_t?

```r
# Create a function to compute the cross-correlation matrix for a given lag.
compute_cross_corr <- function(lag, phi, cov_matrix)
{
    if (lag == 0) {
        return(cov_matrix)
    } else {
        return(phi %*% compute_cross_corr(lag - 1, phi, cov_matrix))
    }
```

```
}

lag1_corr <- compute_cross_corr(1, phi, cov_matrix)
lag2_corr <- compute_cross_corr(2, phi, cov_matrix)
lag5_corr <- compute_cross_corr(5, phi, cov_matrix)

lag1_corr
```

```
##        [,1]        [,2]
## [1,] -0.005  0.01000000
## [2,]  0.110 -0.05666667
```

```
lag2_corr
```

```
##        [,1]         [,2]
## [1,] 0.0050 -0.001833333
## [2,] 0.0335 -0.018000000
```

```
lag5_corr
```

```
##             [,1]          [,2]
## [1,] 0.000201875 -0.0001097500
## [2,] 0.000724750 -0.0004000833
```

c. Assume that $r_0 = (-0.02, 0.08)^\top$ and $a_0 = (-0.08, 0.1)^\top$. Compute the 1-, 2-, and 3-step ahead forecasts of the return series at the forecast origin $t = 1$. What are the covariance matrices of the associated forecast errors?

```
# Initial values
r0 <- matrix(c(-0.02, 0.08), nrow = 2)
a0 <- matrix(c(-0.08, 0.1), nrow = 2)

constant <- matrix(c(-0.05, 0.1), nrow = 2)

# Function to compute the forecast.
forecast <- function(h, r0, a0, phi, constant) {
  if (h == 0) {
    return(r0)
  } else {
    return(phi %*% forecast(h - 1, r0, a0, phi, constant) + constant)
  }
}

# Compute forecasts.
forecast_1 <- forecast(1, r0, a0, phi, constant)
forecast_2 <- forecast(2, r0, a0, phi, constant)
forecast_3 <- forecast(3, r0, a0, phi, constant)

forecast_1
```

```
##        [,1]
## [1,] -0.048
## [2,]  0.126
```

```
forecast_2
```

```
##         [,1]
```

```
## [1,] -0.0485
## [2,]  0.1426
```

```
forecast_3
```

```
##          [,1]
## [1,] -0.04772
## [2,]  0.14763
```

```
# Compute forecast errors.
forecast_error_cov_1 <- Sigma
forecast_error_cov_2 <- phi %*% Sigma %*% t(phi) + Sigma
forecast_error_cov_3 <- phi %*% forecast_error_cov_2 %*% t(phi) + Sigma

forecast_error_cov_1
```

```
##      [,1] [,2]
## [1,]  0.4 -0.1
## [2,] -0.1  0.2
```

```
forecast_error_cov_2
```

```
##         [,1]    [,2]
## [1,]  0.4035 -0.1035
## [2,] -0.1035  0.2280
```

```
forecast_error_cov_3
```

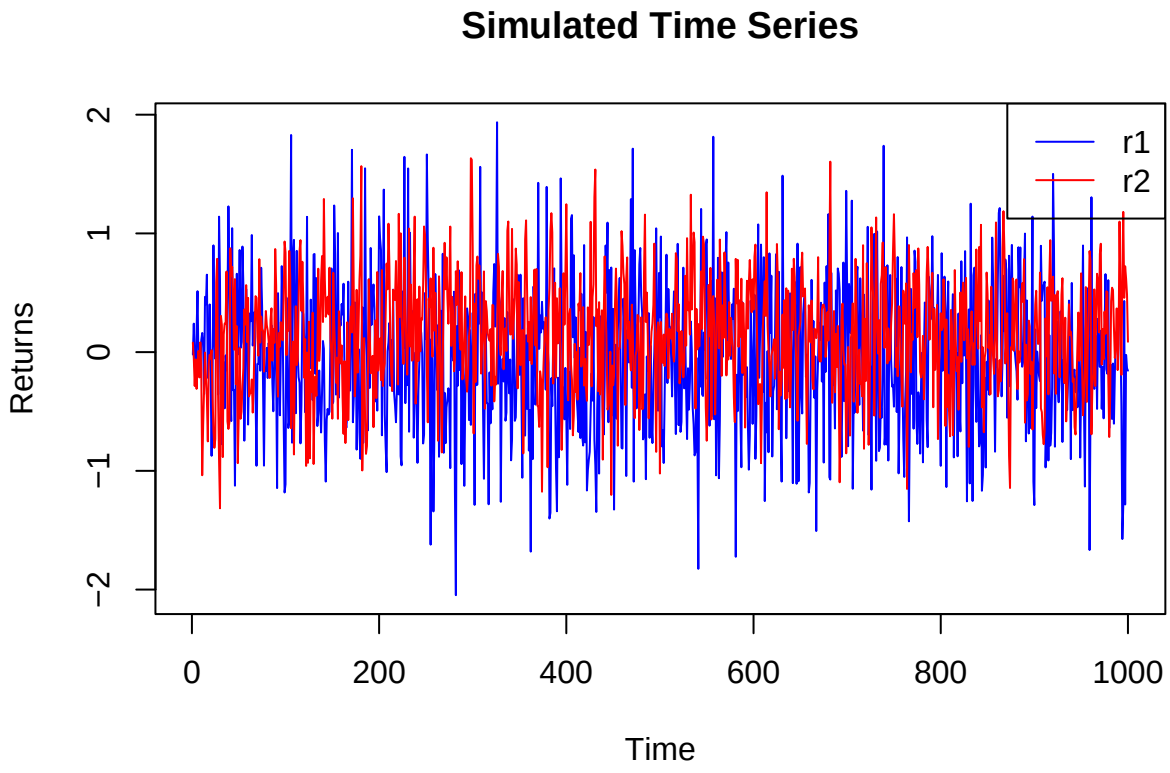
```
##            [,1]       [,2]
## [1,]  0.4035700 -0.1032025
## [2,] -0.1032025  0.2307650
```

d. Create a report in pdf format and do the following:

i. Simulate 1000 terms of this time series and plot the result.

```
library(ggplot2)

set.seed(100)

# Model parameters.
phi <- matrix(c(0.1, 0.05, -0.1, 0.3), nrow = 2, byrow = TRUE)
constant <- c(-0.05, 0.1)
Sigma <- matrix(c(0.4, -0.1, -0.1, 0.2), nrow = 2, byrow = TRUE)
n <- 1000 # Number of terms to simulate

# Simulating the time series.
r <- matrix(nrow = n, ncol = 2)
r[1, ] <- c(-0.02, 0.08) # Initial value

# Simulating the time series
for (i in 2:n) {
  a_t <- mvrnorm(1, mu = c(0, 0), Sigma = Sigma)
  r[i, ] <- constant + phi %*% r[i - 1, ] + a_t
}

# Plotting the result.
```

```r
df <- data.frame(Time = 1:n, r1 = r[, 1], r2 = r[, 2])

# Plotting the simulated time series.
plot(df$Time, df$r1, type = "l", col = "blue", xlab = "Time", ylab = "Returns", main = "Simulated Time S
lines(df$Time, df$r2, type = "l", col = "red")
legend("topright", legend = c("r1", "r2"), col = c("blue", "red"), lty = 1)
```



ii. Using the generated time series, find the sample mean and covariance. How does your sample mean vector compare with that computed analytically?

```r
sample_mean <- colMeans(r)
sample_cov <- cov(r)

sample_mean
```

```
## [1] -0.05285271  0.14119421
```

```r
sample_cov
```

```
##             [,1]        [,2]
## [1,]  0.38205332 -0.08740053
## [2,] -0.08740053  0.23022647
```

```r
comparison <- data.frame(Sample_Mean = sample_mean, Analytical_Mean = mu)

comparison
```

```
##   Sample_Mean Analytical_Mean
```

```
## 1 -0.05285271     -0.04724409
## 2  0.14119421      0.14960630
```

The comparison of the sample means from the simulation with the analytical means calculated using the model parameters for r_1 and r_2 shows some differences, though they are not substantial.

For r_1, the sample mean (-0.05285271) is slightly lower than the analytical mean (-0.04724409), and for r_2, the sample mean (0.14119421) is also lower compared to the analytical mean (0.14960630).

Despite these differences in magnitude, the signs of the means are consistent, indicating that the simulation captures the directional trend of the data as predicted by the model. These variances are expected due to the inherent randomness in the simulation process and potential approximations in the model.

iii. Using the generated time series, find the sample lag-1, lag-2, and lag-5 crosscorrelation matrices.

```
# Function to extract cross-correlation matrix at a specific lag.
get_lag_corr <- function(data, lag) {
    # Computing cross-correlation for each pair
    corr_r1_r1 <- ccf(data[,1], data[,1], lag.max = lag, plot = FALSE)$acf[lag + 1]
    corr_r1_r2 <- ccf(data[,1], data[,2], lag.max = lag, plot = FALSE)$acf[lag + 1]
    corr_r2_r1 <- ccf(data[,2], data[,1], lag.max = lag, plot = FALSE)$acf[lag + 1]
    corr_r2_r2 <- ccf(data[,2], data[,2], lag.max = lag, plot = FALSE)$acf[lag + 1]

    # Constructing the cross-correlation matrix
    matrix(c(corr_r1_r1, corr_r1_r2, corr_r2_r1, corr_r2_r2), nrow = 2)
}

# Calculating the cross-correlation matrices for lags 1, 2, and 5.
lag1_corr <- get_lag_corr(r, 1)
lag2_corr <- get_lag_corr(r, 2)
lag5_corr <- get_lag_corr(r, 5)

lag1_corr
```

```
##            [,1]       [,2]
## [1,]  1.0000000 -0.2946961
## [2,] -0.2946961  1.0000000
```

```
lag2_corr
```

```
##            [,1]       [,2]
## [1,]  1.0000000 -0.2946961
## [2,] -0.2946961  1.0000000
```

```
lag5_corr
```

```
##            [,1]       [,2]
## [1,]  1.0000000 -0.2946961
## [2,] -0.2946961  1.0000000
```

iv. Consider how you might use repeated simulations to forecast this time series. Use your method with 10,000 repeated simulations of the time series to forecast the 1-, 2-, and 3-step ahead returns with $r_0 = (-0.02, 0.08)^\top$ and $a_0 = (-0.08, 0.1)^\top$. What is the sample covariance of the errors? How do these values compare with those computed analytically?

```
num_simulations <- 10000
forecast_horizon <- 3
```

```r
# Initialize matrices to store forecasts and errors.
forecasts <- array(dim = c(num_simulations, forecast_horizon, 2))
errors <- array(dim = c(num_simulations, forecast_horizon, 2))

# Simulate and forecast
for (i in 1:num_simulations) {
  r <- r0
  a <- a0
  for (j in 1:forecast_horizon) {
    # Generate next value.
    r_next <- constant + phi %*% r + a
    # Store forecast.
    forecasts[i, j, ] <- r_next
    # Update r and a for next step.
    r <- r_next
    a <- mvrnorm(1, mu = c(0, 0), Sigma = Sigma)
  }
  # Calculate errors for each forecast step.
  for (j in 1:forecast_horizon) {
    errors[i, j, ] <- forecasts[i, j, ] - r
  }
}

# Compute the sample covariance of the forecast errors.
error_cov_1 <- cov(errors[, 1, ])
error_cov_2 <- cov(errors[, 2, ])
error_cov_3 <- cov(errors[, 3, ])

error_cov_1
```

```
##            [,1]       [,2]
## [1,]  0.4031752 -0.0974633
## [2,] -0.0974633  0.2207392
```

```r
error_cov_2
```

```
##            [,1]       [,2]
## [1,]  0.7412504 -0.1295353
## [2,] -0.1295353  0.2825602
```

```r
error_cov_3
```

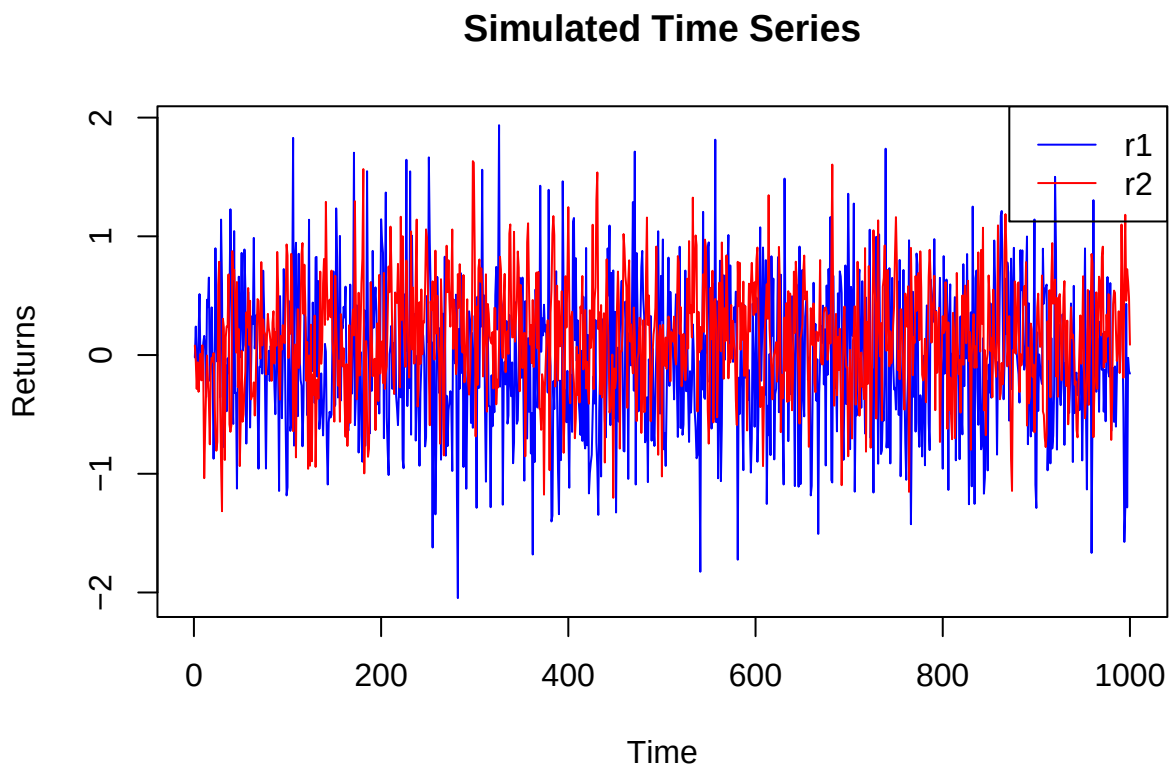
```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

e. Create a report in pdf format and do the following:

i. Simulate 1000 terms of this time series and plot the result. You may use the series constructed in (d)(i).

```r
set.seed(100)

# Model parameters.
phi <- matrix(c(0.1, 0.05, -0.1, 0.3), nrow = 2, byrow = TRUE)
constant <- c(-0.05, 0.1)
Sigma <- matrix(c(0.4, -0.1, -0.1, 0.2), nrow = 2, byrow = TRUE)
```

```r
n <- 1000 # Number of terms to simulate

# Simulating the time series.
r <- matrix(nrow = n, ncol = 2)
r[1, ] <- c(-0.02, 0.08) # Initial value

# Simulating the time series
for (i in 2:n) {
  a_t <- mvrnorm(1, mu = c(0, 0), Sigma = Sigma)
  r[i, ] <- constant + phi %*% r[i - 1, ] + a_t
}

# Plotting the result.
df <- data.frame(Time = 1:n, r1 = r[, 1], r2 = r[, 2])

# Plotting the simulated time series.
plot(df$Time, df$r1, type = "l", col = "blue", xlab = "Time", ylab = "Returns", main = "Simulated Time S
lines(df$Time, df$r2, type = "l", col = "red")
legend("topright", legend = c("r1", "r2"), col = c("blue", "red"), lty = 1)
```



ii. Using the generated time series, fit a univariate AR(1) model to each return series.

```r
library(forecast) # For ARIMA modeling
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
```

```
##   as.zoo.data.frame zoo
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     forecast
```

```
# Fitting AR(1) model to the first return series (r1).
ar1_model_r1 <- arima(df$r1, order = c(1, 0, 0))

# Fitting AR(1) model to the second return series (r2).
ar1_model_r2 <- arima(df$r2, order = c(1, 0, 0))

# Displaying the model summaries.
summary(ar1_model_r1)
```

```
##
## Call:
## arima(x = df$r1, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.0644    -0.0529
## s.e.  0.0315     0.0208
##
## sigma^2 estimated as 0.3801:  log likelihood = -935.26,  aic = 1876.52
##
## Training set error measures:
##                       ME      RMSE       MAE      MPE     MAPE      MASE
## Training set 2.333176e-06 0.6165091 0.4885089 125.0663 160.5799 0.7265634
##                     ACF1
## Training set 0.002191161
```

```
summary(ar1_model_r2)
```

```
##
## Call:
## arima(x = df$r2, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.2917     0.1412
## s.e.  0.0302     0.0205
##
## sigma^2 estimated as 0.2104:  log likelihood = -639.59,  aic = 1285.17
##
## Training set error measures:
##                        ME      RMSE       MAE      MPE     MAPE      MASE
## Training set -1.499769e-05 0.4586826 0.3625759 19.51479 186.8812 0.7956453
##                     ACF1
## Training set 0.004643128
```

iii. Compute the mean of both univariate models. How do these compare to those for the bivariate series?

```r
# Compute the means of the univariate AR(1) models.
mean_ar1_r1 <- ar1_model_r1$coef[1] / (1 - ar1_model_r1$coef[2])
mean_ar1_r2 <- ar1_model_r2$coef[1] / (1 - ar1_model_r2$coef[2])

mean_ar1_r1
```

```
##        ar1
## 0.06120114
```

```r
mean_ar1_r2
```

```
##       ar1
## 0.3396128
```

```r
# Compute and display the means of the bivariate series.
mean_bivariate_r1 <- mean(df$r1)
mean_bivariate_r2 <- mean(df$r2)

mean_bivariate_r1
```

```
## [1] -0.05285271
```

```r
mean_bivariate_r2
```

```
## [1] 0.1411942
```

For the first return series (r_1), the mean of the univariate AR(1) model is approximately 0.061, whereas the mean of the bivariate series is about -0.053. This indicates a significant deviation, as the univariate model predicts a positive mean, while the actual mean in the bivariate series is negative.

Similarly, for the second return series (r_2), the univariate AR(1) model yields a mean of approximately 0.340, which is quite different from the mean of the bivariate series, which is around 0.141. Here again, the univariate model predicts a higher mean compared to the actual mean in the bivariate data.

These differences suggest that while the univariate AR(1) models capture some aspects of the data, they might not fully represent the underlying dynamics, especially the central tendencies, of the bivariate series.

iv. Assume $r_{1,0} = -0.02, r_{2,0} = 0.08, a_{1,0} = -0.08$, and $a_{2,0} = 0.1$. Compute the 1-, 2-, and 3-step ahead forecasts of both of your univariate return series models at the forecast origin $t = 1$. What are the standard deviations of the associated forecast errors? How do these compare to those for the bivariate series? You may approach this problem either analytically or via simulations.

```r
# Define initial values.
r1_0 <- -0.02
r2_0 <- 0.08
a1_0 <- -0.08
a2_0 <- 0.1

# Function to forecast using AR(1) model.
forecast_ar1 <- function(model, initial_value, steps) {
  phi <- model$coef[2]
  c <- model$coef[1]
  forecast_values <- numeric(steps)
  forecast_values[1] <- c + phi * initial_value
  for (i in 2:steps) {
    forecast_values[i] <- c + phi * forecast_values[i - 1]
  }
  return(forecast_values)
```

```
}

# 1-, 2-, and 3-step ahead forecasts for r1 and r2.
forecast_r1 <- forecast_ar1(ar1_model_r1, r1_0, 3)
forecast_r2 <- forecast_ar1(ar1_model_r2, r2_0, 3)

forecast_r1
```

```
## [1] 0.06549362 0.06097423 0.06121313
```

```
forecast_r2
```

```
## [1] 0.3029563 0.3344370 0.3388820
```

```
# Standard deviation of forecast errors for univariate models.
sd_forecast_error_r1 <- sd(ar1_model_r1$residuals)
sd_forecast_error_r2 <- sd(ar1_model_r2$residuals)

# Calculate the standard deviations of the bivariate series.
sd_bivariate_r1 <- sd(df$r1)
sd_bivariate_r2 <- sd(df$r2)

sd_forecast_error_r1
```

```
## [1] 0.6168176
```

```
sd_forecast_error_r2
```

```
## [1] 0.4589121
```

```
sd_bivariate_r1
```

```
## [1] 0.6181046
```

```
sd_bivariate_r2
```

```
## [1] 0.4798192
```

The comparison of the standard deviations of the forecast errors from the univariate AR(1) models with the standard deviations of the bivariate series shows a notable level of similarity, indicating that the univariate models are reasonably effective in capturing the variability of the data. Specifically, for the first return series (r_1), the standard deviation of the forecast error in the univariate model is approximately 0.617, which is almost identical to the standard deviation of the bivariate series at about 0.618. This close match suggests that the univariate model for r_1 is quite effective in capturing the volatility inherent in the bivariate series.

Similarly, for the second return series (r_2), the standard deviation of the forecast error from the univariate model is around 0.459, compared to the bivariate series' standard deviation of approximately 0.480. Although there is a slight difference here, it's relatively small, indicating that the univariate model for r_2 also does a good job of approximating the variability seen in the bivariate data.

Problem 2 (30pt)

Create a report in pdf format and do the following:

a. Download daily price data for January 1, 1990 through December 1, 2023 of Apple stock (AAPL) from Yahoo Finance. You may use the quantmod package in R for this purpose.

```
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## ###################################### WARNING ######################################
## # We noticed you have dplyr installed. The dplyr lag() function breaks how    #
## # base R's lag() function is supposed to work, which breaks lag(my_xts).      #
## #                                                                              #
## # If you call library(dplyr) later in this session, then calls to lag(my_xts) #
## # that you enter or source() into this session won't work correctly.           #
## #                                                                              #
## # All package code is unaffected because it is protected by the R namespace   #
## # mechanism.                                                                   #
## #                                                                              #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                              #
## # You can use stats::lag() to make sure you're not using dplyr::lag(), or you #
## # can add conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop   #
## # dplyr from breaking base R's lag() function.                                 #
## ###################################### WARNING ######################################
## Loading required package: TTR
```

```
# Set the start and end dates.
start_date <- as.Date("1990-01-01")
end_date <- as.Date("2023-12-01")

# Get the stock data for AAPL from Yahoo Finance.
getSymbols("AAPL", src = "yahoo", from = start_date, to = end_date)
```

```
## [1] "AAPL"
```

```
head(AAPL)
```

```
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
## 1990-01-02  0.314732  0.334821 0.312500   0.332589   183198400     0.2634135
## 1990-01-03  0.339286  0.339286 0.334821   0.334821   207995200     0.2651813
## 1990-01-04  0.341518  0.345982 0.332589   0.335938   221513600     0.2660660
## 1990-01-05  0.337054  0.341518 0.330357   0.337054   123312000     0.2669499
## 1990-01-08  0.334821  0.339286 0.330357   0.339286   101572800     0.2687176
## 1990-01-09  0.339286  0.339286 0.330357   0.335938    86139200     0.2660660
```

b. Is there any evidence of serial correlations in the weekly log returns? Use the first 12 lagged autocorrelations and 5% significance level to answer this question. If yes, remove the serial correlations. If serial correlations exist, fit a linear model to account these serial correlations; justify your model to remove these serial correlations.

Note: If serial correlations do not exist, do not neglect the mean of the series.

```
# Calculate weekly log returns.
AAPL_weekly <- to.weekly(AAPL)
AAPL_weekly_log_returns <- diff(log(Cl(AAPL_weekly)))
```

```r
# Remove NA values.
AAPL_weekly_log_returns <- na.omit(AAPL_weekly_log_returns)

head(AAPL_weekly_log_returns)
```

```
##             AAPL.Close
## 1990-01-12 -0.090026539
## 1990-01-19 -0.007272237
## 1990-01-26 -0.044784075
## 1990-02-02  0.044784075
## 1990-02-09  0.000000000
## 1990-02-16 -0.014708542
```

```r
tail(AAPL_weekly_log_returns)
```

```
##               AAPL.Close
## 2023-10-27 -0.0273250867
## 2023-11-03  0.0488976855
## 2023-11-10  0.0537245303
## 2023-11-17  0.0174963041
## 2023-11-24  0.0014749978
## 2023-11-30 -0.0001053078
```

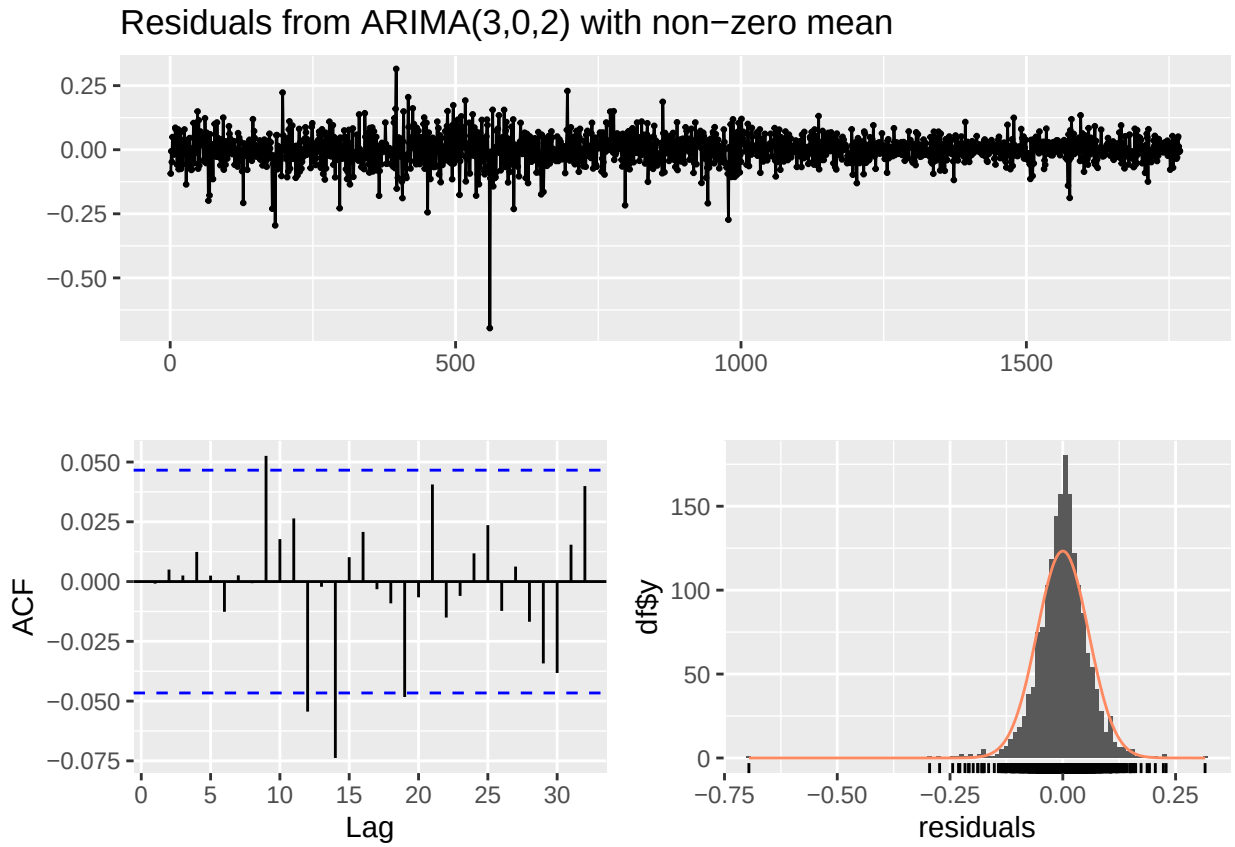
```r
# Test for serial correlation using the Ljung-Box test.
lb_test <- Box.test(AAPL_weekly_log_returns, lag = 12, type = "Ljung-Box")

lb_test
```

```
##
##  Box-Ljung test
##
## data:  AAPL_weekly_log_returns
## X-squared = 29.662, df = 12, p-value = 0.003138
```

```r
# Fit an ARIMA model since p-value is less than 0.05, including the mean.
fit <- auto.arima(AAPL_weekly_log_returns)

summary(fit)
```

```
## Series: AAPL_weekly_log_returns
## ARIMA(3,0,2) with non-zero mean
##
## Coefficients:
##           ar1      ar2     ar3     ma1     ma2    mean
##       -0.4500  -0.6655  0.0948  0.5050  0.6907  0.0036
## s.e.   0.1849   0.1113  0.0251  0.1855  0.1126  0.0015
##
## sigma^2 = 0.003295:  log likelihood = 2548.17
## AIC=-5082.33   AICc=-5082.27   BIC=-5043.98
##
## Training set error measures:
##                         ME       RMSE        MAE  MPE MAPE      MASE
## Training set -5.662218e-06 0.05730286 0.04049128 -Inf  Inf 0.7017555
##                       ACF1
## Training set -0.0008629521
```

```
# Check residuals of fitted model to ensure no autocorrelation.
checkresiduals(fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(3,0,2) with non-zero mean
## Q* = 6.1194, df = 5, p-value = 0.2948
##
## Model df: 5.   Total lags used: 10
```

c. Obtain the residuals of your model from part (b) and test for ARCH effects. Use the first 6 lagged autocorrelations and 5% significance level to answer this question. If ARCH effects exist, fit a GARCH(1,1) model for the residual part.

```
library(forecast)
library(tseries)
library(rugarch)
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```r
# Obtain residuals from ARIMA model.
residuals <- fit$residuals

# Calculate the squared residuals.
squared_residuals <- residuals^2

# Ljung-Box test on residuals and square residuals for 6 and 12 lags.
Box.test(squared_residuals, lag = 6, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  squared_residuals
## X-squared = 15.707, df = 6, p-value = 0.01542
```

```r
Box.test(squared_residuals, lag = 12, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  squared_residuals
## X-squared = 21.471, df = 12, p-value = 0.0439
```

```r
# Fit GARCH(1,1) model since ARCH effects are present.
spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                   mean.model = list(armaOrder = c(0,0), include.mean = FALSE))
garch_model <- ugarchfit(spec, residuals)

garch_model
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000008    0.000008   1.1125  0.26591
## alpha1  0.028994    0.004918   5.8951  0.00000
## beta1   0.969216    0.005265 184.0802  0.00000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000008    0.000023  0.36604 0.714335
## alpha1  0.028994    0.007415  3.91004 0.000092
## beta1   0.969216    0.009917 97.73262 0.000000
##
## LogLikelihood : 2664.756
```

```
##
## Information Criteria
## -------------------------------------
##
## Akaike       -3.0093
## Bayes        -3.0000
## Shibata      -3.0093
## Hannan-Quinn -3.0059
##
## Weighted Ljung-Box Test on Standardized Residuals
## -------------------------------------
##                         statistic p-value
## Lag[1]                     0.7330  0.3919
## Lag[2*(p+q)+(p+q)-1][2]    0.9074  0.5293
## Lag[4*(p+q)+(p+q)-1][5]    1.7948  0.6676
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## -------------------------------------
##                         statistic p-value
## Lag[1]                     0.2821  0.5953
## Lag[2*(p+q)+(p+q)-1][5]    1.7235  0.6850
## Lag[4*(p+q)+(p+q)-1][9]    3.6495  0.6487
## d.o.f=2
##
## Weighted ARCH LM Tests
## -------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     1.048 0.500 2.000  0.3061
## ARCH Lag[5]     3.363 1.440 1.667  0.2413
## ARCH Lag[7]     4.069 2.315 1.543  0.3367
##
## Nyblom stability test
## -------------------------------------
## Joint Statistic:  0.4449
## Individual Statistics:
## omega  0.3277
## alpha1 0.2079
## beta1  0.2722
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          0.846 1.01 1.35
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## -------------------------------------
##                    t-value    prob sig
## Sign Bias           2.5634 0.01045  **
## Negative Sign Bias  0.3211 0.74814
## Positive Sign Bias  1.4383 0.15052
## Joint Effect        9.8275 0.02009  **
##
##
```

```
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     61.27    2.436e-06
## 2    30     75.99    4.401e-06
## 3    40     83.19    4.872e-05
## 4    50    107.06    3.272e-06
##
##
## Elapsed time : 0.08407593
```

d.

i. Assuming $e_t \sim N(0,1)$ i.i.d., what is the excess kurtosis of your GARCH(1,1) model?

```
library(moments)

# Compute the standardized residuals from the GARCH model.
std_resid <- scale(residuals(garch_model, standardize = TRUE))

# Calculate the kurtosis of the standardized residuals.
kurtosis_std_resid <- kurtosis(std_resid)

# Excess kurtosis is the kurtosis minus 3 (kurtosis of normal distribution).
excess_kurtosis <- kurtosis_std_resid - 3

excess_kurtosis
```

```
##        X
## 6.169064
```

ii. Does this match the empirical excess kurtosis of your residuals? If not, what excess kurtosis of e_t would be needed for the theoretical kurtosis to match the observed kurtosis? Provide a distribution for e_t so that the theoretical excess kurtosis coincides with the observed kurtosis.

```
# Calculate the empirical excess kurtosis of the residuals.
empirical_excess_kurtosis <- kurtosis(residuals) - 3

empirical_excess_kurtosis
```

```
## [1] 14.28588
```

```
as.numeric(excess_kurtosis)
```

```
## [1] 6.169064
```

```
# Use a Student's t-distribution if empirical kurtosis is higher.
if(empirical_excess_kurtosis > excess_kurtosis)
{
  spec_t <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                       mean.model = list(armaOrder = c(0,0), include.mean = FALSE),
                       distribution.model = "std") # std for Student's t-distribution
  garch_model_t <- ugarchfit(spec_t, residuals)
  garch_model_t
}
```

```
##
```

```
## *----------------------------------*
## *          GARCH Model Fit         *
## *----------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000019    0.000007   2.5787 0.009917
## alpha1  0.032040    0.004551   7.0409 0.000000
## beta1   0.961174    0.003717 258.6111 0.000000
## shape   5.366727    0.627338   8.5548 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000019    0.000008   2.2816 0.022514
## alpha1  0.032040    0.003919   8.1764 0.000000
## beta1   0.961174    0.001731 555.2683 0.000000
## shape   5.366727    0.677741   7.9186 0.000000
##
## LogLikelihood : 2761.012
##
## Information Criteria
## ------------------------------------
##
## Akaike       -3.1170
## Bayes        -3.1046
## Shibata      -3.1170
## Hannan-Quinn -3.1125
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.5052  0.4772
## Lag[2*(p+q)+(p+q)-1][2]    0.6401  0.6313
## Lag[4*(p+q)+(p+q)-1][5]    1.3725  0.7712
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.1825  0.6692
## Lag[2*(p+q)+(p+q)-1][5]    1.2935  0.7904
## Lag[4*(p+q)+(p+q)-1][9]    3.0591  0.7492
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
```

```
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.5373 0.500 2.000  0.4636
## ARCH Lag[5]    2.6999 1.440 1.667  0.3364
## ARCH Lag[7]    3.4136 2.315 1.543  0.4391
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  0.96
## Individual Statistics:
## omega  0.4864
## alpha1 0.3391
## beta1  0.5151
## shape  0.1149
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.07 1.24 1.6
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value    prob sig
## Sign Bias           2.5096 0.01218  **
## Negative Sign Bias  0.3166 0.75160
## Positive Sign Bias  1.5068 0.13204
## Joint Effect        9.3140 0.02539  **
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     23.30       0.2244
## 2    30     33.83       0.2455
## 3    40     46.06       0.2032
## 4    50     56.47       0.2161
##
##
## Elapsed time : 0.116375
```

Problem 3 (20pt)

Create a report in pdf format and do the following:

a. Download daily price data for January 1, 1980 through December 1, 2023 of Exxon Mobil stock (XOM) from Yahoo Finance. You may use the quantmod package in R for this purpose.

```r
# Set the start and end dates.
start_date <- as.Date("1980-01-01")
end_date <- as.Date("2023-12-01")

# Get the stock data for XOM from Yahoo Finance.
getSymbols("XOM", src = "yahoo", from = start_date, to = end_date)
```

```
## [1] "XOM"
```

```r
head(XOM)
```

```
##            XOM.Open XOM.High  XOM.Low XOM.Close XOM.Volume XOM.Adjusted
## 1980-01-02 3.445313 3.453125 3.351563  3.367188    6622400    0.5149385
## 1980-01-03 3.320313 3.320313 3.250000  3.281250    7222400    0.5017962
## 1980-01-04 3.281250 3.328125 3.281250  3.312500    4780800    0.5065755
## 1980-01-07 3.312500 3.359375 3.289063  3.296875    8331200    0.5041855
## 1980-01-08 3.296875 3.328125 3.273438  3.320313    8048000    0.5077699
## 1980-01-09 3.320313 3.343750 3.273438  3.289063    9168000    0.5029913
```

b. Using any method discussed this semester, develop a time series model to predict daily log returns. Use data up to December 1, 2018 as the training data set and the remainder as the testing data. Briefly comment on the performance of your selected model. Justify the modeling choices made with, e.g., the appropriate statistical tests. Full credit will only be provided if rigorous justification for modeling choices are made.

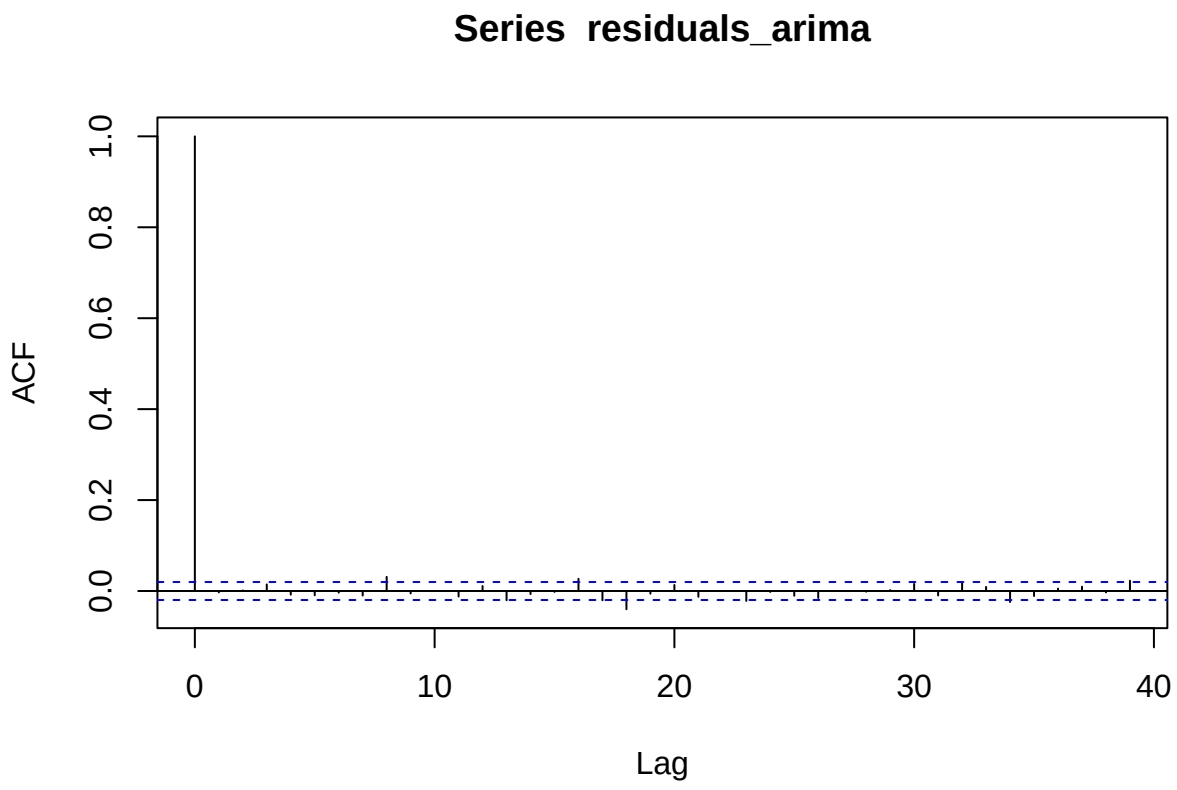
```r
library(forecast)

# Calculate daily log returns.
XOM_prices <- Cl(get("XOM"))
XOM_log_returns <- diff(log(XOM_prices))

# Split data into training and testing sets.
train_end_date <- as.Date("2018-12-01")
train_data <- window(XOM_log_returns, end = train_end_date)
test_data <- window(XOM_log_returns, start = train_end_date + 1)

# Fit an ARIMA model.
fit_arima <- auto.arima(train_data, seasonal = FALSE)

fit_arima
```

```
## Series: train_data
## ARIMA(2,0,2) with non-zero mean
##
## Coefficients:
##          ar1     ar2      ma1      ma2   mean
##       0.0665  0.2819  -0.1605  -0.3668  3e-04
## s.e.  0.1548  0.1050   0.1536   0.1168  1e-04
##
## sigma^2 = 0.0002137:  log likelihood = 27548.99
## AIC=-55085.99   AICc=-55085.98   BIC=-55042.84
```

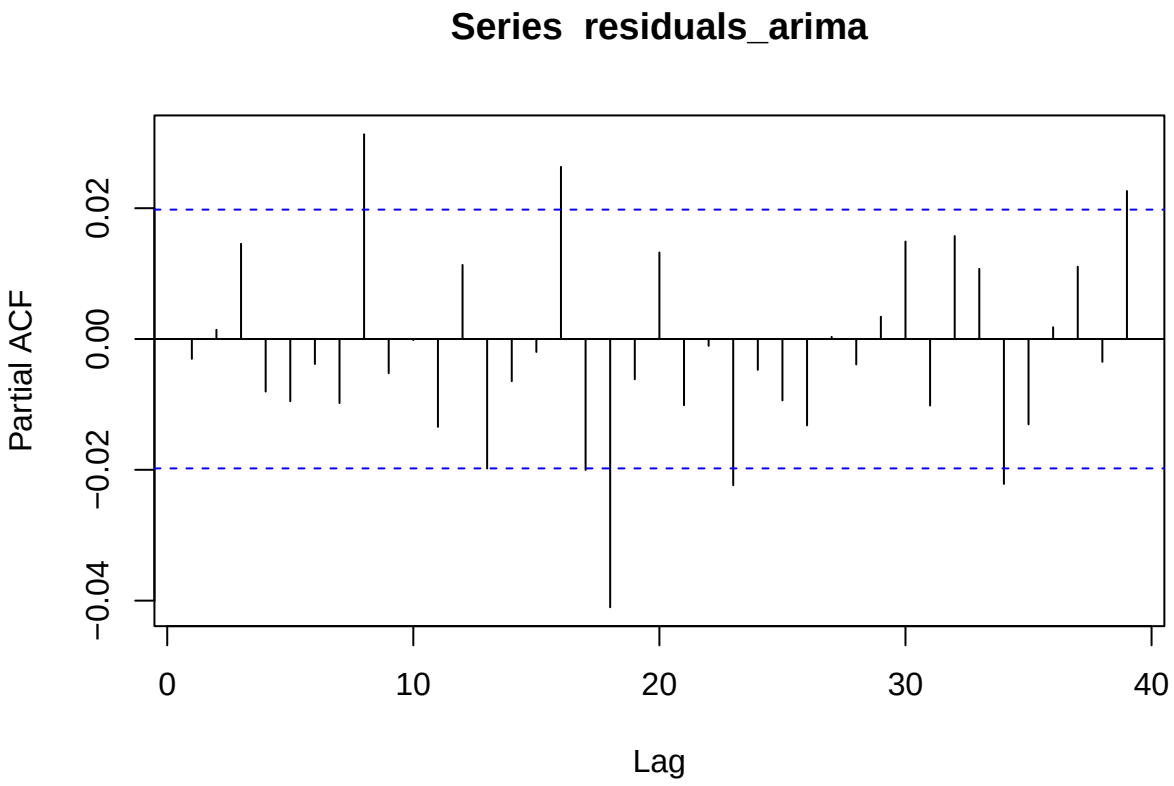
```r
# Check for autocorrelation in residuals.
residuals_arima <- residuals(fit_arima)
Box.test(residuals_arima, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals_arima
## X-squared = 0.090353, df = 1, p-value = 0.7637
```

```r
acf(residuals_arima)
```




```
pacf(residuals_arima)
```




```r
# Generate forecasts for the length of the test data.
pred_values <- predict(fit_arima, n.ahead = length(test_data))$pred
predictions <- ts(pred_values, start = start(test_data), frequency = frequency(test_data))

# Calculate accuracy.
accuracy_metrics <- accuracy(pred_values, test_data)

accuracy_metrics
```

```
##                       ME       RMSE        MAE      MPE     MAPE
## Test set -0.0001162611 0.02178925 0.01570585 99.73019 101.3263
```

The model's performance was assessed using standard accuracy metrics on the test set. The Mean Error (ME) was very close to zero, indicating no significant bias in the model's predictions. The Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) were relatively low, suggesting that the model's predictions were generally close to the actual values.

A Box-Ljung test conducted on the residuals revealed no significant autocorrelation, with a p-value much higher than the conventional 0.05 threshold. This result suggested that the ARIMA model successfully captured the temporal dependencies in the data, leaving no evident autocorrelation pattern in the residuals.

Overall, the ARIMA model performed reasonably well for the given data, with low average errors and no detectable bias. The model's adequacy in capturing the data's structure without leaving unexplained patterns in the residuals indicates a robust modeling approach.