Lecture 3, September 15, 2025

Estimation of ARIMA models

Marta Boczon

Department of Economics

Copnehagen Business School

mbo.eco@cbs.dk

In [90]:
```
#install.packages("quantmod")
#install.packages("fredr")
#install.packages("ggfortify")
#install.packages('urca')
#install.packages("tseries")
#install.packages("forecast")
#install.packages("dynlm")
#install.packages("stargazer")
#install.packages("dLagM")
#install.packages("pracma")
```

Exercise 1

Test each of the five series below example a) through e) for a unit root using the ADF test.

In [58]:
```
##################################################################################
# Example A
T = 200                     # Set the number of time periods (length of the series)
ex.a = rep(0, T)            # Create an empty vector of length T to store the simulated series
set.seed(123)               # Fix the random seed for reproducibility
e = rnorm(T, 0, 0.2)        # Generate T random errors from N(0, 0.2^2)
beta0 = 40                  # Set the intercept (starting level of the series)
beta1 = 0.4                 # Set the slope (deterministic trend)
t = 1:T                     # Create a time index from 1 to T

# Loop over each time period and compute the value of y_t
for (i in 1:T) {
  ex.a[i] = beta0 + beta1 * t[i] + e[i]
}
##################################################################################
# Example B
T = 200                     # Set the number of time periods (length of the series)
ex.b = rep(0, T)            # Create an empty vector of length T to store the simulated series
set.seed(123)               # Fix the random seed for reproducibility
e = rnorm(T, 0, 0.2)        # Generate T random errors from N(0, 0.2^2)
beta0 = 0.8                 # Constant term (drift component)
beta1 = 0.4                 # Autoregressive coefficient (effect of past y on current y)
beta2 = 0.1                 # Coefficient on time trend (deterministic component)
t = 1:T                     # Create a time index from 1 to T

# Loop from the 2nd observation onwards
# Each value depends on:
#   - a constant (beta0)
#   - the lagged value (beta1 * ex.b[i-1]) → autoregressive component
#   - a deterministic time trend (beta2 * t[i])
#   - a random shock (e[i])
for (i in 2:T) {
  ex.b[i] = beta0 + beta1 * ex.b[i-1] + beta2 * t[i] + e[i]
}
##################################################################################
# Example C
T = 200                     # Set the number of time periods (length of the series)
ex.c = rep(0, T)            # Create an empty vector of length T to store the simulated series
set.seed(123)               # Fix the random seed for reproducibility
e = rnorm(T, 0, 0.2)        # Generate T random errors from N(0, 0.2^2)

# Loop from the 2nd observation onwards
# Each value is yesterday’s value plus a random shock (random walk)
for (t in 2:T) {
  ex.c[t] = ex.c[t-1] + e[t]
}
##################################################################################
# Example D
T = 200                     # Set the number of time periods (length of the series)
ex.d = rep(0, T)            # Create an empty vector of length T to store the simulated series
beta0 = 0.5                 # Constant term (drift) that shifts the process upward over time
set.seed(123)               # Fix the random seed for reproducibility
e = rnorm(T, 0, 0.2)        # Generate T random errors from N(0, 0.2^2)

# Loop from the 2nd observation onwards
```

```r
# Each value equals yesterday's value + a constant drift + a random shock
for (t in 2:T) {
  ex.d[t] = beta0 + ex.d[t-1] + e[t]
}
##################################################################################
# Example E
T = 200                         # Set the number of time periods (length of the series)
ex.e = rep(0, T)                # Create an empty vector of length T to store the simulated series
beta0 = 0.5                     # Constant drift term (shifts the process upward each step)
beta1 = 0.9                     # Coefficient on deterministic trend (linear trend over time)
set.seed(123)                   # Fix the random seed for reproducibility
e = rnorm(T, 0, 0.2)            # Generate T random errors from N(0, 0.2^2)
t = 1:T                         # Define time index from 1 to T

# Loop from the 2nd observation onwards
# Each value = yesterday's value + constant drift + time trend + random shock
for (t in 2:T) {
  ex.e[t] = beta0 + beta1 * t + ex.e[t-1] + e[t]
}
```

Solution to Exercise 1

In [61]:
```r
library(urca)
summary(ur.df(ex.a, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.45413 -0.12172 -0.01179  0.12125  0.59979 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 46.73937    6.16594   7.580 1.61e-12 ***
z.lag.1     -1.16065    0.15648  -7.417 4.18e-12 ***
tt           0.46412    0.06259   7.416 4.22e-12 ***
z.diff.lag1  0.10125    0.13637   0.742   0.4587    
z.diff.lag2  0.02452    0.10662   0.230   0.8184    
z.diff.lag3  0.12064    0.07268   1.660   0.0986 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1875 on 185 degrees of freedom
Multiple R-squared:  0.5508,	Adjusted R-squared:  0.5387 
F-statistic: 45.37 on 5 and 185 DF,  p-value: < 2.2e-16


Value of test-statistic is: -7.4174 88.3098 27.612 

Critical values for test statistics: 
      1pct  5pct 10pct
tau3 -3.99 -3.43 -3.13
phi2  6.22  4.75  4.07
phi3  8.43  6.49  5.47
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")
- **τ_3 = −7.42** vs **1% crit = −3.99** → **Reject H_0 (unit root)**.
- **ϕ_2 = 88.31** vs **1% crit = 6.22** → deterministic terms (constant + trend) are **jointly significant**.
- **ϕ_3 = 27.61** vs **1% crit = 8.43** → the **trend term is significant**.

Conclusion

The series **does not have a unit root** under a regression that includes a deterministic trend. It is **trend-stationary**: stationary **around a linear trend**.

Next steps: either (i) keep a linear time trend in the mean equation when modeling (e.g., ARMA with trend), or (ii) detrend the series via OLS and model the residuals.

In [62]:
```r
library(urca)
summary(ur.df(ex.b, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.45530 -0.11462 -0.01743  0.12388  0.60814 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.020e+00  1.142e-01   8.931 4.19e-16 ***
z.lag.1     -7.112e-01  1.098e-01  -6.475 8.28e-10 ***
tt           1.184e-01  1.830e-02   6.470 8.55e-10 ***
z.diff.lag1  4.501e-02  1.028e-01   0.438   0.6621    
z.diff.lag2  3.248e-05  8.736e-02   0.000   0.9997    
z.diff.lag3  1.254e-01  7.282e-02   1.722   0.0868 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1881 on 185 degrees of freedom
Multiple R-squared:  0.3553,	Adjusted R-squared:  0.3379 
F-statistic: 20.39 on 5 and 185 DF,  p-value: 3.458e-16


Value of test-statistic is: -6.4754 53.1713 21.042 

Critical values for test statistics: 
      1pct  5pct 10pct
tau3 -3.99 -3.43 -3.13
phi2  6.22  4.75  4.07
phi3  8.43  6.49  5.47
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")
- **τ_3 = −6.48** vs **1% crit = −3.99** → **Reject H_0 (unit root)**.
- **ϕ_2 = 53.17** vs **1% crit = 6.22** → deterministic terms (constant + trend) are **jointly significant**.
- **ϕ_3 = 21.04** vs **1% crit = 8.43** → the **trend term is significant**.

Conclusion

The series **does not have a unit root** when a deterministic trend is included. It is **trend-stationary**: stationary **around a linear trend**.

Next steps: either (i) keep a linear time trend in the mean equation when modeling (e.g., ARMA with trend), or (ii) detrend the series via OLS and model the residuals.

In [63]:
```
library(urca)
summary(ur.df(ex.c, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max
-0.43810 -0.10702 -0.01389  0.11725  0.59344

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.0565919  0.0349227   1.620   0.1068
z.lag.1     -0.0600566  0.0259327  -2.316   0.0217 *
tt          -0.0003077  0.0002560  -1.202   0.2309
z.diff.lag  -0.0451358  0.0732050  -0.617   0.5383
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1868 on 187 degrees of freedom
Multiple R-squared:  0.03477,	Adjusted R-squared:  0.01929
F-statistic: 2.246 on 3 and 187 DF,  p-value: 0.08445


Value of test-statistic is: -2.3159 1.9111 2.838

Critical values for test statistics:
      1pct  5pct 10pct
tau3 -3.99 -3.43 -3.13
phi2  6.22  4.75  4.07
phi3  8.43  6.49  5.47
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")
- **ϕ_2 = 1.911** vs **crit (6.22, 4.75, 4.07)** → constant **not needed**.
- **ϕ_3 = 2.838** vs **crit (8.43, 6.49, 5.47)** → trend **not needed**.

Conclusion

Both **ϕ-tests** are below critical values → the deterministic components (constant and trend) are **jointly insignificant**.
Therefore, the trend-spec regression is **over-specified**.

Next step: re-run the ADF with no deterministic terms, i.e. `type = "none"`.

In [64]:
```
library(urca)
summary(ur.df(ex.c, type='none', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression none


Call:
lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max
-0.43347 -0.09684  0.00740  0.13018  0.61349

Coefficients:
           Estimate Std. Error t value Pr(>|t|)
z.lag.1    -0.03255    0.01895  -1.718   0.0875 .
z.diff.lag -0.05828    0.07276  -0.801   0.4242
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1871 on 189 degrees of freedom
Multiple R-squared:  0.02073,	Adjusted R-squared:  0.01037
F-statistic:     2 on 2 and 189 DF,  p-value: 0.1382


Value of test-statistic is: -1.7176

Critical values for test statistics:
      1pct  5pct 10pct
tau1 -2.58 -1.95 -1.62
```

Interpretation of the Unit Root Test

Test statistics: $\tau_1 = -1.718$
Critical values: 1% = −2.58, 5% = −1.95, 10% = −1.62

Interpretation

- At **5%**, −1.718 is **not** more negative than −1.95 ⇒ **fail to reject** the unit-root null.
- At **10%**, −1.718 < −1.62 ⇒ **reject** at 10% only (very weak evidence).

Conclusion

Using the conventional 5% level, the series **has a unit root** under the no-constant/no-trend specification.
→ Treat the series as **I(1)** (difference once before modeling), unless you are willing to accept the weaker 10% evidence.

In [65]:
```r
library(urca)
summary(ur.df(ex.d, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max
-0.43810 -0.10702 -0.01389  0.11725  0.59344

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.54913    0.04643  11.827   <2e-16 ***
z.lag.1     -0.06006    0.02593  -2.316   0.0217 *
tt           0.02972    0.01290   2.305   0.0223 *
z.diff.lag  -0.04514    0.07320  -0.617   0.5383
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1868 on 187 degrees of freedom
Multiple R-squared:  0.03477,	Adjusted R-squared:  0.01929
F-statistic: 2.246 on 3 and 187 DF,  p-value: 0.08445


Value of test-statistic is: -2.3159 66.6523 2.838

Critical values for test statistics:
      1pct  5pct 10pct
tau3 -3.99 -3.43 -3.13
phi2  6.22  4.75  4.07
phi3  8.43  6.49  5.47
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")
- **τ_3 = −2.316** vs **crit: 1% = −3.99, 5% = −3.43, 10% = −3.13**
- **ϕ_2 = 66.652** vs **crit: 1% = 6.22, 5% = 4.75, 10% = 4.07**
- **ϕ_3 = 2.838** vs **crit: 1% = 8.43, 5% = 6.49, 10% = 5.47**

Interpretation

- **Unit root:** −2.316 is *not* more negative than any critical value ⇒ **fail to reject** the unit-root null under the trend specification.
- **Deterministic terms:**
  • **ϕ_2** is far above critical values ⇒ deterministic part is **not jointly zero** (keep a deterministic component).
  • **ϕ_3** is below all critical values ⇒ **no evidence of a trend** term.

Conclusion

The trend regression is **over-specified**. The maintained specification should be **with a constant but no trend** (`type = "drift"`). Re-run the ADF with `type = "drift"` and base the unit-root decision on **τ_2**.

In [66]:
```r
library(urca)
summary(ur.df(ex.d, type='drift', lags=8, selectlags="AIC"))
```

```
###############################################
# 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
-0.47070 -0.12064 -0.01958  0.11823  0.64100

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.5497046  0.0469608  11.706   <2e-16 ***
z.lag.1     -0.0002974  0.0004986  -0.596    0.552
z.diff.lag  -0.0749641  0.0728735  -1.029    0.305
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1889 on 188 degrees of freedom
Multiple R-squared:  0.007355,	Adjusted R-squared:  -0.003205
F-statistic: 0.6965 on 2 and 188 DF,  p-value: 0.4996


Value of test-statistic is: -0.5965 95.1401

Critical values for test statistics:
      1pct  5pct 10pct
tau2 -3.46 -2.88 -2.57
phi1  6.52  4.63  3.81
```

Interpretation of the Unit Root Test

Test statistics (type = "drift")

- **Test statistic = −0.5965** (unit-root test with constant)
- **ϕ_1 = 95.14** (test that the constant = 0) **Critical values (1%, 5%, 10%):**
- Test statistic: −3.46, −2.88, −2.57
- ϕ_1: 6.52, 4.63, 3.81

Interpretation

- **Unit root:** −0.5965 is not more negative than any critical value ⇒ **fail to reject** the unit-root null.
- **Deterministic term:** $\phi_1 \gg$ critical values ⇒ the **constant (drift) is significant** and should be retained.

Conclusion

Under the **drift** specification, the series is **non-stationary (has a unit root)** but includes a **nonzero drift**.
Next step: treat the series as **I(1)** — difference once (and, if appropriate, log first), then re-test the differenced series (usually with `type = "drift"`).

In [67]:
```
library(urca)
summary(ur.df(ex.e, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.47063 -0.12097 -0.02013  0.11830  0.64065 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.448e+00  6.137e-02  23.594   <2e-16 ***
z.lag.1     -1.524e-07  1.120e-05  -0.014    0.989    
tt           9.674e-01  6.577e-02  14.708   <2e-16 ***
z.diff.lag  -7.504e-02  7.308e-02  -1.027    0.306    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1894 on 187 degrees of freedom
Multiple R-squared:      1,	Adjusted R-squared:      1 
F-statistic: 4.368e+06 on 3 and 187 DF,  p-value: < 2.2e-16


Value of test-statistic is: -0.0136 389.8745 108.1773 

Critical values for test statistics: 
      1pct  5pct 10pct
tau3 -3.99 -3.43 -3.13
phi2  6.22  4.75  4.07
phi3  8.43  6.49  5.47
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")
- **Test statistic = −0.0136** (unit-root test with trend)
- **ϕ_2 = 389.87** (constant + trend jointly = 0?)
- **ϕ_3 = 108.18** (trend = 0?) **Critical values (1% / 5% / 10%):**
- Test statistic: −3.99 / −3.43 / −3.13
- ϕ_2: 6.22 / 4.75 / 4.07
- ϕ_3: 8.43 / 6.49 / 5.47

Interpretation

- **Unit root:** The test statistic (−0.0136) is far above the critical values ⇒ **fail to reject** a unit root.
- **Deterministic terms:** ϕ_2 and ϕ_3 ≫ critical values ⇒ **keep both a constant and a linear trend** in levels.

Conclusion

The series is **non-stationary with a unit root** and includes a **deterministic trend**.
After differencing, **do not switch to** `type = "drift"` by default. Use **`type = "trend"`** in the ADF on Δy_t, or detrend Δy_t on t and test the residuals with `type = "none"` (or `type = "drift"` if the detrended mean ≠ 0).

Roadmap

Box–Jenkins Model Selection

Identification stage:

- Visually examine the time plot (detect outliers, missing values, structural breaks, non-stationarity)
- Address non-stationarity
- Examine Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF), and identify potential models

Estimation and diagnostic stage:

- Estimate all potential models
- Select the best model(s) using suitable criterion/criteria
- Check ACF and PACF of the residuals

Application stage:

- Use the model(s) to forecast

Empirical Example

Today's lecture will focus on learning by doing with a real data example. We will walk step by step through the process of analyzing a time series, starting from the very beginning and moving all the way to model comparison. Specifically, we will:

1. Import the dataset into our software.
2. Create a time series object.
3. Plot the data to get an initial look at patterns.
4. Test whether the series has a unit root.
5. Apply treatments for non-stationarity if necessary and re-test.
6. Plot the data, the autocorrelation function (ACF), and the partial autocorrelation function (PACF).
7. Estimate various ARMA models.
8. Compare the competing models using information criteria.
9. Export results.

Step 1

Import the data

For today's example, we'll be working with data from the **Federal Reserve Bank of St. Louis**, often referred to as **FRED** (Federal Reserve Economic Data).

What is FRED?

FRED is one of the most widely used databases in economics and finance. It's maintained by the St. Louis Fed and provides free access to thousands of economic time series from around the world.

Why do we use it?

- It's reliable and regularly updated.
- It covers a huge variety of economic indicators (GDP, inflation, unemployment, exchange rates, stock market data, etc.).
- It's easy to access in many formats (**Excel**, **CSV**, **R**, **Stata**, etc.).

For us today:

We'll use FRED as our data source to illustrate the steps of time series analysis. The nice thing about using FRED data is that the same tools and workflow you'll practice today can be applied to almost any economic variable you're interested in.

- **Data:** Canadian GDP (Real Gross Domestic Product for Canada)
- **Source:** International Monetary Fund via FRED®
- **Frequency:** Quarterly
- **Units:** Millions of Domestic Currency
- **Seasonal adjustment:** Seasonally Adjusted

https://fred.stlouisfed.org/series/NGDPRSAXDCCAQ

In [3]:
```
# Import the dataset from a CSV file downloaded from FRED
data =read.csv("NGDPRSAXDCCAQ.csv")

# Display the first six rows of the dataset
# This helps us check the structure of the data and the column names
head(data)

# Display the last six rows of the dataset
# This lets us confirm how recent the observations are and the overall time span
tail(data)
```

A data.frame: 6 × 2

	observation_date	NGDPRSAXDCCAQ
	<chr>	<dbl>
1	1961-01-01	90980.8
2	1961-04-01	93284.5
3	1961-07-01	95554.5
4	1961-10-01	96854.5
5	1962-01-01	99431.8
6	1962-04-01	100222.0

A data.frame: 6 × 2

	observation_date	NGDPRSAXDCCAQ
	<chr>	<dbl>
252	2023-10-01	599156.5
253	2024-01-01	600215.5
254	2024-04-01	604005.5
255	2024-07-01	607606.3
256	2024-10-01	610704.3
257	2025-01-01	614059.8

Conclusion

We see that our data runs from 1961Q1 to 2025Q1.

Step 2

Create a time series object

In [60]:
```
# Convert the second column of the dataset (the actual GDP values) into a time series object
# 'frequency = 4' specifies that the data are quarterly
# 'start = c(1961, 1)' indicates that the series begins in the first quarter of 1961
tsdata = ts(data[, 2], frequency = 4, start = c(1961, 1))

# A time series object in R (created with the ts() function)
# is a special data structure designed to store and work
# with data observed at regular time intervals,
# such as monthly inflation, quarterly GDP, or daily stock prices.
# Regular intervals: A time series object knows the frequency of the data:
# (e.g., 4 for quarterly, 12 for monthly, 365 for daily).
# Start: It records when the series begins (e.g., 1961 Q1) and when it ends.
# Built-in functionality: Many R functions
# (like plotting, computing ACF/PACF, or estimating ARMA models)
# are designed to work directly with ts objects.
# That’s why it’s important to convert raw data into this format.

# Print the time series object to verify it was created correctly
tsdata
```

A Time Series: 65 × 4

	Qtr1	Qtr2	Qtr3	Qtr4
1961	90980.8	93284.5	95554.5	96854.5
1962	99431.8	100222.0	101503.8	103495.5
1963	104151.8	105666.5	106469.5	109801.5
1964	112550.3	112971.8	114177.5	114750.3
1965	118400.5	119484.5	121152.0	124121.3
1966	126536.5	129268.0	129206.3	130807.5
1967	130512.8	133231.8	133651.8	134234.0
1968	135378.3	139017.8	140954.0	142905.8
1969	144862.3	145458.3	147072.0	149150.3
1970	150072.0	150084.5	151637.8	151868.0
1971	151086.8	155498.3	159795.5	161542.0
1972	161001.5	165565.8	166057.0	169957.5
1973	174319.5	175982.5	176800.8	180061.8
1974	181552.0	183414.3	183449.5	182493.8
1975	181499.5	183828.0	187354.8	189522.8
1976	192825.5	196983.8	197893.3	198213.0
1977	201253.3	202280.3	203469.3	206693.5
1978	208478.5	209897.5	211855.5	213864.3
1979	215570.5	218430.3	220031.5	221507.8
1980	223045.8	222948.3	222763.0	225845.3
1981	230710.8	233233.3	231136.3	230073.3
1982	227440.3	224821.8	222828.0	220771.8
1983	224379.8	228806.3	231376.8	234253.3
1984	238310.5	242935.3	243979.8	247787.5
1985	251594.5	252403.8	255608.0	259505.0
1986	259373.5	260864.3	261192.3	259301.5
1987	265295.0	268693.5	272798.8	276323.0
1988	280349.0	282836.8	282878.8	284820.0
1989	288111.0	289231.8	290276.0	289698.3
1990	292484.3	291310.8	289223.0	286647.8
1991	282524.0	283880.0	284256.5	284756.0
1992	284957.0	285281.8	286827.0	288461.3
1993	290292.8	292937.0	295738.5	297003.5
1994	301382.8	305782.5	309702.8	311952.5
1995	314818.8	314952.5	315368.8	316641.5
1996	317101.0	319358.5	322040.8	324592.3
1997	328941.5	332643.3	336586.8	339839.5
1998	344638.0	344820.5	347962.5	352677.3
1999	359045.0	362006.3	367680.5	372831.5
2000	378669.0	383125.8	387053.0	387818.5
2001	390001.5	391228.0	390953.0	393297.8
2002	399048.5	401406.3	404877.3	407101.0
2003	409366.0	408772.3	410300.5	413121.0
2004	416125.0	421055.5	426033.8	429108.5
2005	430611.3	433723.5	438979.8	443339.5
2006	446939.0	447162.5	448421.3	450207.0
2007	453080.3	457455.8	459169.8	459773.3
2008	460118.3	461794.8	465589.5	460187.3
2009	449802.8	444950.8	446940.8	452133.8
2010	457611.5	460007.8	463228.0	468424.5
2011	471992.0	472848.8	479318.5	483127.8
2012	483432.8	485004.3	485668.0	486667.5

	Qtr1	Qtr2	Qtr3	Qtr4
2013	491023.8	493851.5	497912.5	503123.5
2014	503973.8	508567.8	513442.8	516991.5
2015	514113.0	512709.5	514545.5	514886.5
2016	517933.0	515353.0	520701.3	523622.5
2017	530210.8	535791.5	535959.8	538679.0
2018	544941.3	549192.0	551962.3	553262.5
2019	554594.0	560665.5	562233.3	563838.5
2020	552844.0	492031.5	536725.3	546807.0
2021	556122.3	555470.5	566660.5	576806.0
2022	583642.0	589642.8	592680.3	591868.3
2023	599885.3	600741.0	599386.0	599156.5
2024	600215.5	604005.5	607606.3	610704.3
2025	614059.8			

In [10]:

```
# Calculate the number of observations in the time series object
nd = length(tsdata)

# Print a message along with the number of observations
cat("There are", nd, "observations in our data set.")
```

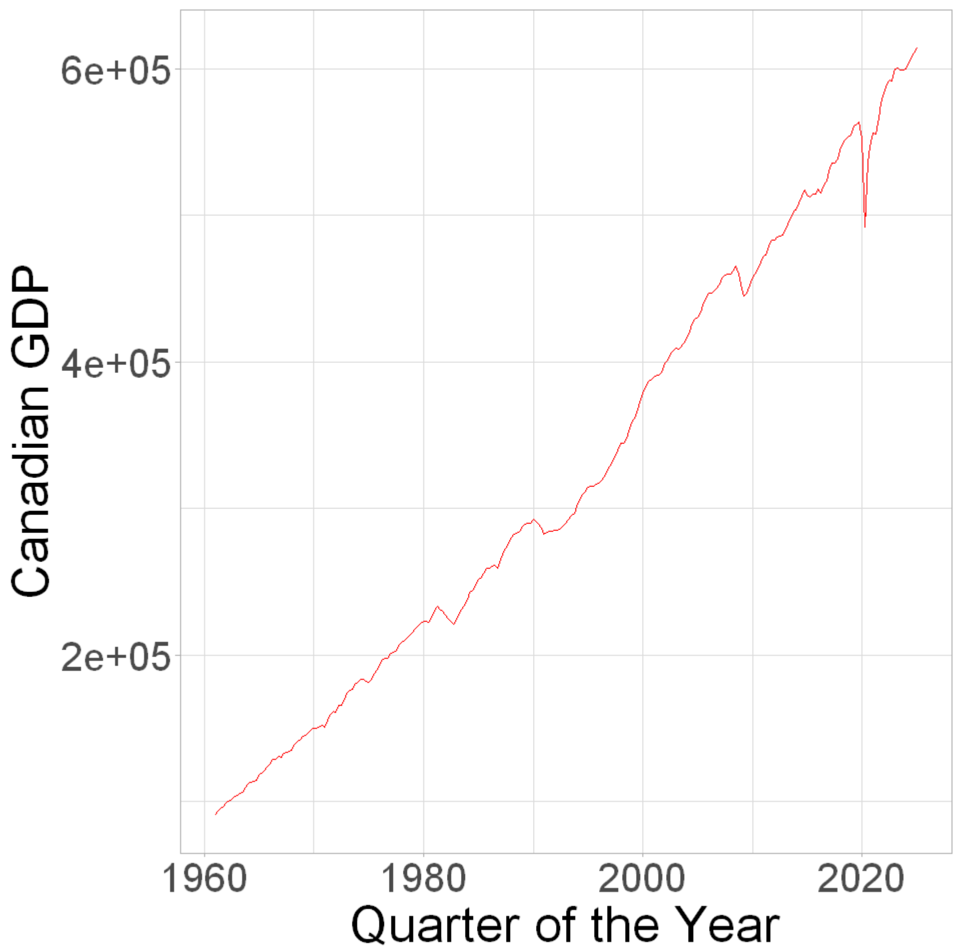
```
There are 257 observations in our data set.
```

Step 3

Plot the data

In [13]:

```
# Load the ggplot2 package for plotting
library(ggplot2)

# Load ggfortify, which allows autoplot() to work directly with time series objects
library(ggfortify)

# Set default figure size for plots
options(repr.plot.width = 8, repr.plot.height = 8)

# Create a plot of the time series object 'tsdata' in red
fig = autoplot(tsdata, colour = 'red')

# Customize the plot appearance
fig = fig +
  theme(aspect.ratio = 1) +   # Keep a square aspect ratio
  theme_light() +             # Use a light background theme
  theme(plot.margin = ggplot2::margin(0.2, 0.2, 0.2, 0.2, "cm")) + # Adjust margins
  theme(text = element_text(size = 30)) + # Increase font size for readability
  labs(x = "Quarter of the Year") +          # Label the x-axis
  labs(y = "Canadian GDP")    # Label the y-axis

# Display the plot
fig
```



Step 4

Test for a unit root

In [15]:
```
# Load the urca package, which provides functions for unit root and cointegration tests
library(urca)

# Perform an Augmented Dickey-Fuller (ADF) test for a unit root
# - tsdata: the time series we are testing
# - type = "trend": includes both a constant and a deterministic trend in the test regression
# - lags = 8: sets the maximum number of lags to include
# - selectlags = "AIC": automatically selects the optimal number of lags
# based on the Akaike Information Criterion
summary(ur.df(tsdata, type = 'trend', lags = 8, selectlags = "AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
   Min     1Q Median     3Q    Max
-64305  -1193    364   1811  33356

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 4713.60919 1542.64517   3.056   0.0025 **
z.lag.1       -0.04977    0.02158  -2.307   0.0219 *
tt           106.55674   44.10408   2.416   0.0164 *
z.diff.lag    -0.09458    0.06383  -1.482   0.1397
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5436 on 244 degrees of freedom
Multiple R-squared:  0.03929,	Adjusted R-squared:  0.02748
F-statistic: 3.326 on 3 and 244 DF,  p-value: 0.02035


Value of test-statistic is: -2.3066 15.1122 3.294

Critical values for test statistics:
      1pct  5pct 10pct
tau3 -3.98 -3.42 -3.13
phi2  6.15  4.71  4.05
phi3  8.34  6.30  5.36
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")

- ϕ_2 = 15.1122 vs 1% crit = 6.15 → Reject H_0 (constant and deterministic trend are jointly zero).
- ϕ_3 = 3.294 vs 1% crit = 8.34 → Do not reject H_0 (deterministic trend = 0).

Conclusion

Because the ϕ-tests give conflicting results, the regression including a deterministic trend is **misspecified**.
We should therefore re-run the test with `type = "drift"` (constant only).

In [16]:
```
library(urca)
summary(ur.df(tsdata, type='drift', lags=8, selectlags="AIC"))
```

```
###############################################
# 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
-64883  -1252    201   1820  34769

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.619e+03  8.683e+02   1.865   0.0634 .
z.lag.1      2.048e-03  2.387e-03   0.858   0.3917
z.diff.lag  -1.200e-01  6.358e-02  -1.888   0.0603 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5489 on 245 degrees of freedom
Multiple R-squared:  0.0163,    Adjusted R-squared:  0.008275
F-statistic:  2.03 on 2 and 245 DF,  p-value: 0.1335


Value of test-statistic is: 0.858 19.3673

Critical values for test statistics:
      1pct  5pct 10pct
tau2 -3.44 -2.87 -2.57
phi1  6.47  4.61  3.79
```

Interpretation of the Unit Root Test

Test statistics (type = "drift")
- **ϕ_1 = 19.3673** vs **1% crit = 6.47** → **Reject H_0** (constant = 0) → **drift specification is appropriate**.
- **τ_2 = 0.858** vs **1% crit = −3.44** → **Do not reject H_0** (unit root).

Conclusion

The **drift specification** is correct, but the ADF test indicates that the series has a **unit root (non-stationary)**.
This means we will need to **difference the series** before proceeding with modeling.

Step 5

Treat for non-stationarity if necessary and re-test

Unit Root and Logarithmic Transformation

- The **ADF test** only checks for unit roots.
- In an **additive random walk** (with or without drift), first differencing removes the unit root:
  - This cures both:
    - the time dependence of the mean, and
    - the time dependence of the variance (since shocks are additive with constant variance). See proofs in Lecture 2.
- But in most **real-world data** (e.g., GDP, stock prices, exchange rates), shocks are **multiplicative** — larger levels come with larger fluctuations.
  - In this case, differencing alone will not fix the problem of variance scaling with levels.
- To address this, we first apply a **natural log transformation**, which turns multiplicative shocks into additive ones.
- Then, taking **first differences of the logged data** removes the unit root *and* yields a series with approximately constant mean and variance.

Stationarity Issues: Multiplicative Shocks vs. Unit Root

Multiplicative Shocks (Heteroskedasticity Due to Scale)

Process:

$$y_t = y_{t-1} \cdot e^{\varepsilon_t}$$

- Each new shock ε_t **scales with the level of** y_{t-1}.
- As y_t grows, the same-size **percentage shock** leads to a bigger **absolute movement**.
- This is why variance appears to **grow with the level** (bigger economy → bigger fluctuations in absolute terms).

Solution: take **logs** (turns multiplicative shocks into additive).

Unit Root (Memory of Past Shocks)

Process:

$$y_t = y_{t-1} + \varepsilon_t$$

- Here shocks are **additive**.
- The problem is not their size, but that they **accumulate forever**.
- Variance grows with time:

$$Var(y_t) = t\sigma^2$$

- Why? Because each ε_t stays in the series forever (no mean reversion).

Solution: take **first differences** (removes the accumulated memory).

Key Distinction

- **Multiplicative shocks** → variance grows with **scale** (big numbers → bigger swings).
- **Unit root** → variance grows with **time** (because past shocks never die out).

That's why for GDP (or prices) we often:

1. Take **logs** (fixes multiplicative scaling).
2. Take **first differences** (fixes unit root memory).

→ Giving **stationary log growth rates**.

In [20]:
```
# Transform the series into (approximate) percentage changes
# - log(tsdata): takes the natural log to convert multiplicative shocks into additive shocks
#                (stabilizes variance across time)
# - diff(...): takes first differences to remove the unit root (non-stationarity in the mean)
# Together: dy is the log-differenced series, which represents GDP growth rates
#           with approximately constant mean and variance
dy = diff(log(tsdata))
```

Detour on Logarithmic Transformation

The growth rate **g** is defined as:

$$g = \frac{y_t - y_{t-1}}{y_{t-1}}$$

Now, let us see if we can approximate **g** by taking the first difference of a natural logarithm of y_t:

$$\Delta \log y_t = \log(y_t) - \log(y_{t-1}) = \log\left(\frac{y_t}{y_{t-1}}\right) = \log\left(\frac{y_t - y_{t-1} + y_{t-1}}{y_{t-1}}\right) = \log\left(\frac{y_t - y_{t-1}}{y_{t-1}} + 1\right) \approx \frac{y_t - y_{t-1}}{y_{t-1}}$$

This approximation holds true because for any small x:

$$\log(1 + x) \approx x$$

Re-Testing

- We start from **drift**, because that's where we left off in the levels.
- Including a deterministic trend now would be inconsistent — it was already rejected at the first stage.

Interpretation

- If the ADF with drift **rejects** → the differenced series (Δy_t) is **stationary**.

- If not → the differenced series still has a **unit root** → difference again (testing for I(2)), still under **drift** as the maintained specification.

In [26]:
```
# Load the urca package, which provides functions for unit root and cointegration tests
library(urca)

# Perform an Augmented Dickey-Fuller (ADF) test on the differenced log series (dy)
# - dy: the log-differenced data, i.e. approximate growth rates
# - type = "trend": includes a constant and a deterministic trend in the test regression
# - lags = 8: sets the maximum number of lags to consider
# - selectlags = "AIC": automatically selects the optimal lag length
# using the Akaike Information Criterion
# The summary output reports the tau and phi statistics for unit root testing
summary(ur.df(dy, type = "drift", lags = 8, selectlags = "AIC"))
```

```
###############################################
# 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
-0.123222 -0.004833 -0.000034  0.004944  0.081984

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.006871   0.001029   6.680  1.6e-10 ***
z.lag.1     -0.964626   0.089884 -10.732  < 2e-16 ***
z.diff.lag  -0.022828   0.063963  -0.357    0.721
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01261 on 244 degrees of freedom
Multiple R-squared:  0.4942,	Adjusted R-squared:   0.49
F-statistic: 119.2 on 2 and 244 DF,  p-value: < 2.2e-16


Value of test-statistic is: -10.7319 57.5879

Critical values for test statistics:
      1pct  5pct 10pct
tau2 -3.44 -2.87 -2.57
phi1  6.47  4.61  3.79
```

Interpretation of the Unit Root Test

Test statistics (type = "drift")
- **ϕ_1 = 57.59** vs **1% crit = 6.47** → **Reject H_0** (constant = 0) → **including a constant (drift) is appropriate**.
- **τ_2 = −10.73** vs **1% crit = −3.44** → **Reject H_0** (unit root).

Conclusion

After first differencing, the series **no longer has a unit root**.
The differenced log series (GDP growth) is **stationary around a constant mean**.

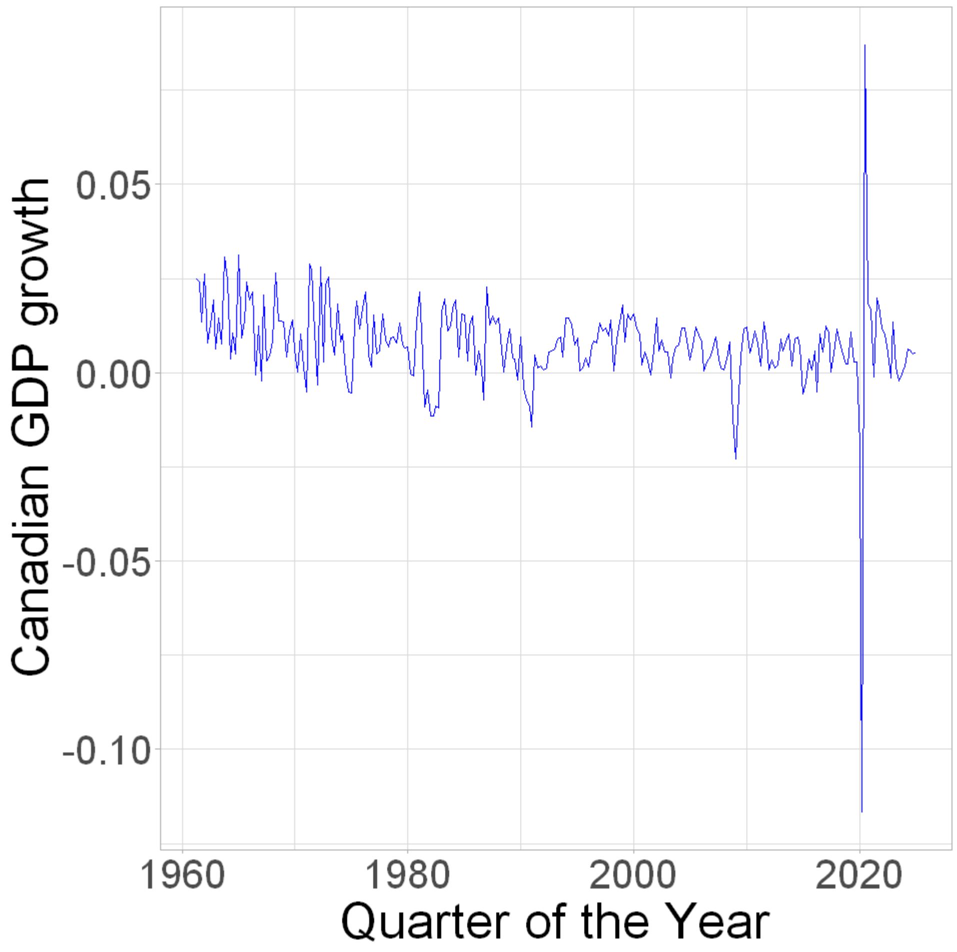
This means the series is now suitable for **ARMA-type modeling**.

Step 6

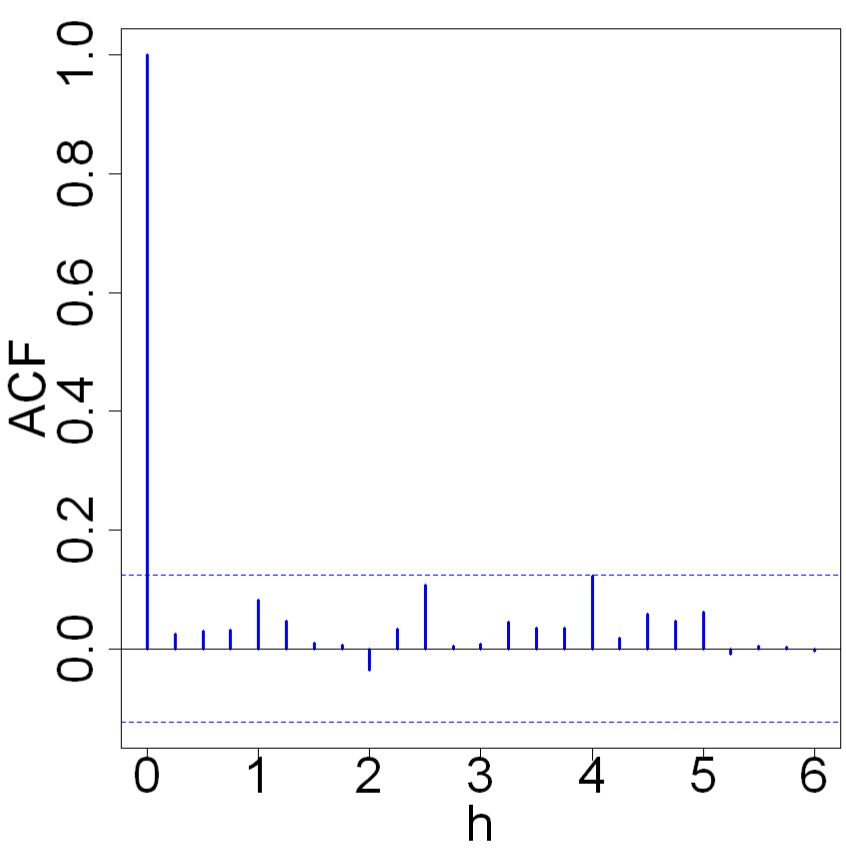
Plot the data, the ACF and PACF

In [51]:
```
# Load the ggplot2 package for plotting
library(ggplot2)

# Load ggfortify, which allows autoplot() to work directly with time series objects
library(ggfortify)

# Set default figure size for plots
options(repr.plot.width = 8, repr.plot.height = 8)

# Create a plot of the time series object 'tsdata' in red
fig = autoplot(dy, colour = 'blue')

# Customize the plot appearance
fig = fig +
```

```
  theme(aspect.ratio = 1) +   # Keep a square aspect ratio
  theme_light() +             # Use a light background theme
  theme(plot.margin = ggplot2::margin(0.2, 0.2, 0.2, 0.2, "cm")) + # Adjust margins
  theme(text = element_text(size = 30)) + # Increase font size for readability
  labs(x = "Quarter of the Year") +           # Label the x-axis
  labs(y = "Canadian GDP growth")    # Label the y-axis

# Display the plot
fig
```

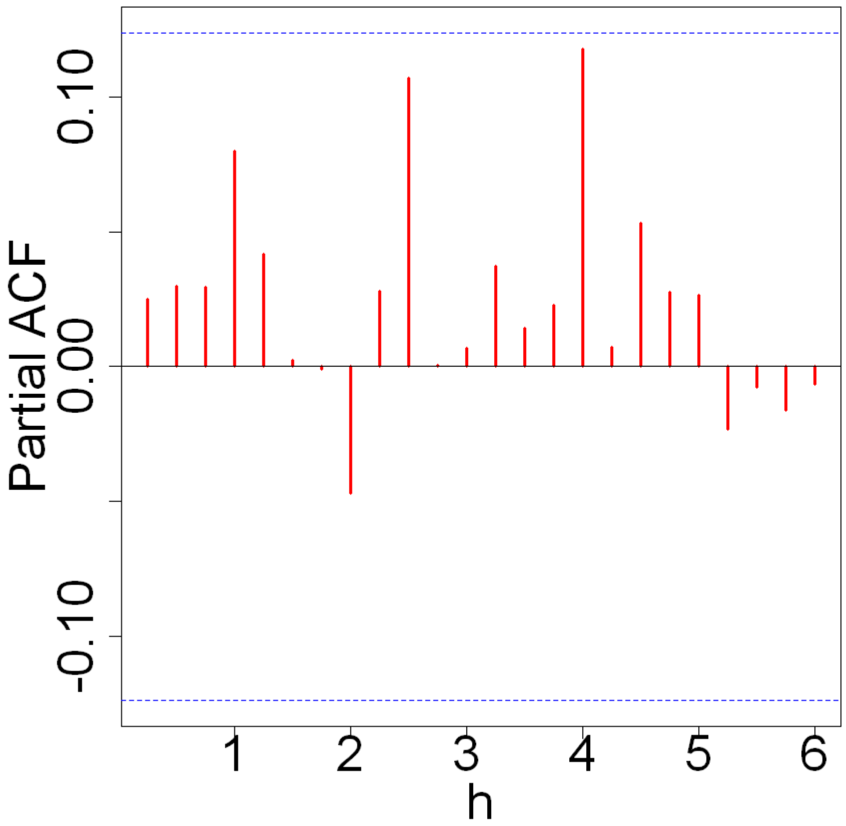


In [12]:
```
# Compute the autocorrelation function (ACF) of dy without plotting
acfun = acf(dy, plot = FALSE)

# Set the size of the plotting window (useful in Jupyter / R notebooks)
options(repr.plot.width = 8, repr.plot.height = 8)

# Adjust margins: bottom, left, top, right
par(mar = c(5, 5, 5, 5))

# Plot the ACF with custom styling
plot(acfun,
     col = "blue",         # bars in blue
     main = "",            # remove the default main title
     cex = 2.5,            # scale size of points (not critical here)
     cex.lab = 2.5,        # enlarge axis labels (xlab, ylab)
     cex.axis = 2.5,       # enlarge tick labels
     xlab = "h",           # label for x-axis (lag h)
     lwd = 3)              # line width for bars
```



In [13]:
```
# Compute the partial autocorrelation function (PACF) of dy without plotting
pacfun = pacf(dy, plot = FALSE)
```

```
# Set the size of the plotting window (useful in Jupyter / R notebooks)
options(repr.plot.width = 8, repr.plot.height = 8)

# Adjust margins: bottom, left, top, right
par(mar = c(5, 5, 5, 5))

# Plot the PACF with custom styling
plot(pacfun,
     col = "red",        # bars in red
     main = "",          # remove the default main title
     cex = 2.5,          # scale size of points (not critical here)
     cex.lab = 2.5,      # enlarge axis labels (xlab, ylab)
     cex.axis = 2.5,     # enlarge tick labels
     xlab = "h",         # label for x-axis (lag h)
     lwd = 3)            # line width for bars
```



Explanation

The ACF does not provide a clear rule for detection of AR processes. The reason behind it is that **indirect autocorrelations** are present in the ACF of any autoregressive process — for example, in an AR(1) there is a correlation between y_t and y_{t-2}.

However, notice that the **partial autocorrelation** between y_t and y_{t-s} eliminates the effects of the intervening values y_{t-1} through y_{t-s+1}. Therefore, in an AR(1) process, the PAC of order 2 is zero.

As such, the PACF provides a convenient rule for detection of a pure AR process: its order is the largest k for which the partial autocorrelation is different from zero.

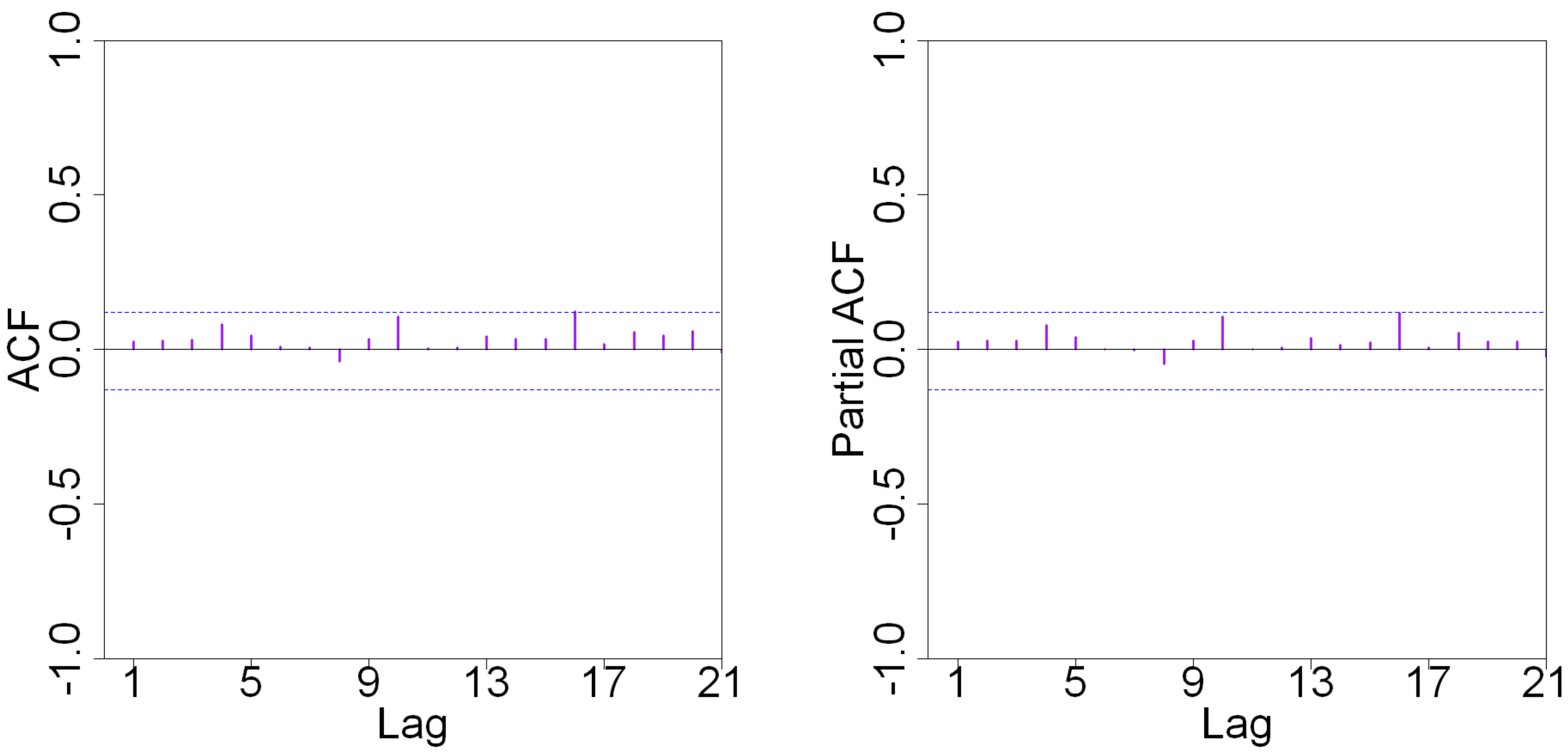
For an **invertible MA process**, the PACF measures the autocorrelations in the corresponding AR(∞) representation. Hence the **slow decay towards zero**.

Takeaway: ACF can help us detect the order of an MA process — the last significant lag of the ACF function is the suggested lag of the MA process.

Takeaway: PACF can help us detect the order of an AR process — the last significant lag of the PACF function is the suggested lag of the AR process.

In [14]:
```
myplot.acf = function(ACFobj) {
    rr = ACFobj$acf[-1]
    kk = length(rr)
    nn = ACFobj$n.used
    plot(seq(kk), rr, type = "h", lwd = 3, yaxs = "i", xaxs = "i",
        ylim = c(floor(min(rr)), 1), xlim = c(0, 21), xlab = "Lag",
        ylab = "ACF", col="purple", main="", cex = 2.5,
         cex.lab=2.5, cex.axis=2.5, cex.main=2.5, cex.sub=2.5, xaxp = c(1, 21, 5))
    abline(h = -1/nn + c(-2, 2)/sqrt(nn), lty = "dashed", col = "blue")
    abline(h = 0)
}

myplot.pacf = function(PACFobj) {
    rr = PACFobj$acf
    kk = length(rr)
    nn = PACFobj$n.used
    plot(seq(kk), rr, type = "h", lwd = 3, yaxs = "i", xaxs = "i",
        ylim = c(floor(min(rr)), 1), xlim = c(0, 21), xlab = "Lag",
        ylab = "Partial ACF", col="purple", main="", cex = 2.5,
```

```
        cex.lab=2.5, cex.axis=2.5, cex.main=2.5, cex.sub=2.5, xaxp = c(1, 21, 5))
    abline(h = -1/nn + c(-2, 2)/sqrt(nn), lty = "dashed", col = "blue")
    abline(h = 0)
}

options(repr.plot.width=16, repr.plot.height=8)
par(mar = c(5, 5, 5, 5), mfrow = c(1,2))
acf.y = acf(dy, plot = FALSE)
myplot.acf(acf.y)

pacf.y = pacf(dy, plot = FALSE)
myplot.pacf(pacf.y)
```



Interpretation of ACF and PACF

ACF (blue plot): after lag 0, all spikes are very small and stay within the confidence bands (dashed lines). This suggests no significant autocorrelation.

PACF (red plot): all spikes are also small, with only a few touching but not clearly exceeding the confidence bounds.

Together, this means:

- There is no evidence of systematic dependence across lags.
- The series `dy` looks like **white noise**: observations are uncorrelated over time.
- In other words, the process does **not show memory** (no persistence or autoregressive structure).

Step 7

Estimate the model

Consider the AR(1) model:

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + e_t, \qquad e_t \sim \text{i.i.d.}(0, \sigma^2).$$

1) Expectation

$$\boxed{E[y_t] = \frac{\alpha_0}{1-\alpha_1}}$$

$\implies \alpha_1 \neq 1.$

2) Variance

$$\boxed{\text{Var}(y_t) = \frac{\sigma^2}{1-\alpha_1^2}}$$

$\implies |\alpha_1| < 1.$

3) Covariance

For lag $k \geq 1$:

$$\boxed{\gamma_k = \alpha_1^{\,k}\,\frac{\sigma^2}{1-\alpha_1^2}}$$

$\Longrightarrow |\alpha_1| < 1.$

Stationarity Condition

$|\alpha_1| < 1$ ensures a constant mean and finite, time-invariant variance and covariance.

Proof

Consider

$$y_t = \phi_0 + \phi_1 y_{t-1} + e_t, \qquad e_t \sim \text{i.i.d.}(0, \sigma^2).$$

1. Expectation

Take expectations:

$$\mathbb{E}[y_t] = \phi_0 + \phi_1 \mathbb{E}[y_{t-1}] + \mathbb{E}[e_t].$$

Since $\mathbb{E}[e_t] = 0$ and under stationarity $\mathbb{E}[y_t] = \mathbb{E}[y_{t-1}] = \mu$:

$$\mu = \phi_0 + \phi_1 \mu \;\Rightarrow\; \mu(1-\phi_1) = \phi_0 \;\Rightarrow\; \mu = \frac{\phi_0}{1-\phi_1}, \quad \phi_1 \neq 1.$$

2. Variance

Take variances:

$$\mathrm{Var}(y_t) = \mathrm{Var}(\phi_0 + \phi_1 y_{t-1} + e_t).$$

Since ϕ_0 is constant and drops out:

$$\mathrm{Var}(y_t) = \mathrm{Var}(\phi_1 y_{t-1} + e_t).$$

Because e_t is independent of y_{t-1}:

$$\mathrm{Var}(y_t) = \phi_1^2\,\mathrm{Var}(y_{t-1}) + \mathrm{Var}(e_t).$$

Under stationarity, $\mathrm{Var}(y_t) = \mathrm{Var}(y_{t-1}) = \gamma_0$, so:

$$\gamma_0 = \phi_1^2 \gamma_0 + \sigma^2 \;\Rightarrow\; \gamma_0(1-\phi_1^2) = \sigma^2 \;\Rightarrow\; \gamma_0 = \frac{\sigma^2}{1-\phi_1^2}, \quad |\phi_1| < 1.$$

3. Covariance

For lag $k \geq 1$:

$$\gamma_k = \mathrm{Cov}(y_t, y_{t-k}) = \mathrm{Cov}(\phi_1 y_{t-1} + e_t,\; y_{t-k}).$$

Because e_t is independent of past values,

$$\gamma_k = \phi_1\,\mathrm{Cov}(y_{t-1}, y_{t-k}) = \phi_1 \gamma_{k-1}.$$

By recursion,

$$\gamma_k = \phi_1^k \gamma_0, \quad k \geq 0.$$

Hence for $|\phi_1| < 1$, the AR(1) process is weakly stationary with

- mean $\mu = \phi_0/(1-\phi_1)$,
- variance $\sigma^2/(1-\phi_1^2)$,
- autocovariance $\gamma_k = \phi_1^k \gamma_0$.

Proof

Consider

$$y_t = \phi_0 + \phi_1 y_{t-1} + e_t, \qquad e_t \sim \text{i.i.d.}(0, \sigma^2).$$

Rewrite:

1. **1st period**

$$y_1 = \alpha_0 + \alpha_1 y_0 + e_1$$

2. **2nd period**

$$y_2 = \alpha_0 + \alpha_1 y_1 + e_2 = \alpha_0 + \alpha_1(\alpha_0 + \alpha_1 y_0 + e_1) + e_2$$

$$y_2 = \alpha_0(1+\alpha_1) + \alpha_1^2 y_0 + \alpha_1 e_1 + e_2$$

3. **3rd period**

$$y_3 = \alpha_0 + \alpha_1 y_2 + e_3$$

$$= \alpha_0 + \alpha_1\left[\alpha_0(1+\alpha_1) + \alpha_1^2 y_0 + \alpha_1 e_1 + e_2\right] + e_3$$

$$= \alpha_0(1+\alpha_1+\alpha_1^2) + \alpha_1^3 y_0 + \alpha_1^2 e_1 + \alpha_1 e_2 + e_3$$

n-th period: By induction, we obtain a general form:

$$y_t = \sum_{j=0}^{t-1} \alpha_0 \alpha_1^j + \alpha_1^t y_0 + \sum_{j=0}^{t-1} \alpha_1^j e_{t-j}$$

Expectation:

$$\mathbb{E}[y_t] = \alpha_0 \sum_{j=0}^{t-1} \alpha_1^j + \alpha_1^t \mathbb{E}[y_0].$$

If $|\alpha_1| < 1$ and $t \to \infty$, then $\alpha_1^t \mathbb{E}[y_0] \to 0$ and the geometric sum converges:

$$\lim_{t\to\infty} \mathbb{E}[y_t] = \frac{\alpha_0}{1-\alpha_1}.$$

Variance:

$$\mathrm{Var}(y_t) = \mathrm{Var}\left(\sum_{j=0}^{t-1} \alpha_1^j e_{t-j}\right).$$

Since e_t are i.i.d.,

$$\mathrm{Var}(y_t) = \sigma^2 \sum_{j=0}^{t-1} (\alpha_1^j)^2.$$

For $|\alpha_1| < 1$ and $t \to \infty$,

$$\lim_{t\to\infty} \mathrm{Var}(y_t) = \frac{\sigma^2}{1-\alpha_1^2}.$$

Covariance:

For lag k:

$$\mathrm{Cov}(y_t, y_{t-k}) = E\left[\left(\sum_{j=0}^{t-1} \alpha_1^j e_{t-j}\right)\left(\sum_{m=0}^{t-k-1} \alpha_1^m e_{t-k-m}\right)\right].$$

Nonzero terms occur only when indices match, giving:

$$\mathrm{Cov}(y_t, y_{t-k}) = \sigma^2 \alpha_1^k \sum_{r=0}^{t-k-1} \alpha_1^{2r}.$$

As $t \to \infty$, this converges to:

$$\mathrm{Cov}(y_t, y_{t-k}) = \alpha_1^k \frac{\sigma^2}{1-\alpha_1^2}.$$

In [47]:
```
library(stargazer)
model.ar1 = arima(dy, order = c(1,0,0))
stargazer(model.ar1, type = "text")
```

```
============================================
                    Dependent variable:
                 ---------------------------
                            dy
--------------------------------------------
ar1                        0.027
                          (0.063)

intercept                 0.007***
                          (0.001)

--------------------------------------------
Observations                256
Log Likelihood            758.550
sigma2                    0.0002
Akaike Inf. Crit.        -1,511.100
============================================
Note:            *p<0.1; **p<0.05; ***p<0.01
```

Estimation Results

- **AR(1) coefficient:** $\hat{\alpha}_1 = 0.027$ (std. error = 0.063) → Statistically insignificant → no meaningful persistence.
- **Intercept (mean):** $\hat{\mu} = 0.007$ (std. error = 0.001, ***p<0.01)
  In R's `arima()` function, the reported **intercept** corresponds to the **unconditional mean**:

$$\mu = E[y_t] = \frac{\alpha_0}{1 - \alpha_1}.$$

→ The expected value of y_t is about **0.7% per period**.

- **Residual variance:** $\hat{\sigma}^2 = 0.0002$
- **Log likelihood:** 758.55
- **AIC:** –1511.10

Expectation
In R's parametrization:

$$E[y_t] = \mu = 0.007.$$

Variance
For stationary AR(1):

$$\mathrm{Var}(y_t) = \frac{\sigma^2}{1 - \alpha_1^2}.$$

Plugging in:

$$\mathrm{Var}(y_t) = \frac{0.0002}{1 - (0.027)^2} \approx 0.000200.$$

First-Order Autocorrelation
By definition:

$$\rho_1 = \alpha_1 \approx 0.027.$$

This is very small and statistically insignificant, so the process shows **almost no persistence**.

Conclusion

- The intercept in R output represents the **mean** ($\mu \approx 0.007$).
- The variance is approximately **0.0002**.
- The AR(1) coefficient is not significant, so the process is essentially **white noise with a positive mean of 0.7% per period**.

Step 8

Compare models using information criteria

Our goal is to select a **stationary** and **parsimonious** model that fits the data well.

In cross-sectional econometrics, the R^2 and adjusted R^2 are usually used for comparing models in terms of fit. But what does $R^2 = 0.49$ really tell us? Can R^2 be used to assess the fit of a time series model?

$$R^2 = \frac{\text{Explained Sum of Squares}}{\text{Total Sum of Squares}} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = \frac{SSE}{SST} = 1 - \frac{SSR}{SST}$$

Model selection criteria account for a **trade-off** between reducing the residual sum of squares and losing degrees of freedom by penalizing the inclusion of too many lags.

Information Criteria

1. **Akaike Information Criterion (AIC)**

$$\text{AIC} = T \ln\left(\frac{SSR}{T}\right) + 2k$$

2. **Bayesian Information Criterion (BIC)**

$$\text{BIC} = T \ln\left(\frac{SSR}{T}\right) + \ln(T)\, k$$

where k denotes the number of regressors (e.g., $p + q$).

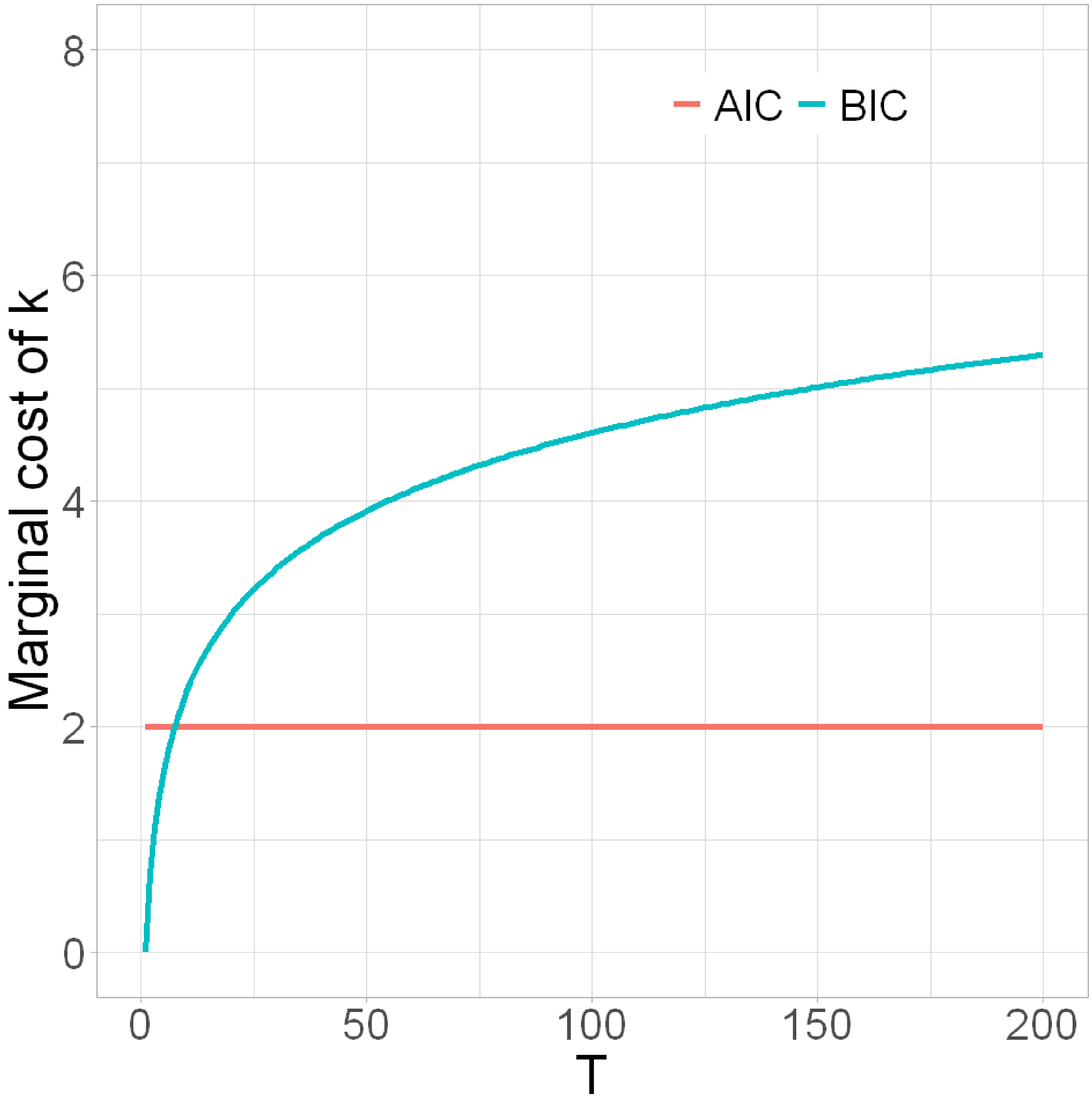
There are several versions of such criteria, but all are functions of the model's residual sum of squares and the number of regressors.

To adequately compare alternative models, T should be kept fixed — *can you see why?*

Decision Rule

Choose the model with the lowest value of the criterion.

Note: Which criterion will provide you with a more parsimonious model?




```
In [53]: # Load the stargazer package, which produces nicely formatted model output
library(stargazer)
```

```r
# Estimate an ARMA(0,0) model on the growth rate series 'dy'
# - order = c(0,0,0) specifies no MA, no differencing, no AR component
#   i.e. dy_t = intercept + error_t
model.arma00 = arima(dy, order = c(0,0,0))

# Print the ARMA(0,0) estimation results in plain text format
# stargazer provides a cleaner regression-like summary than the default arima() output
stargazer(model.arma00, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                               dy
---------------------------------------------
intercept                   0.007***
                             (0.001)

---------------------------------------------
Observations                   256
Log Likelihood               758.456
sigma2                        0.0002
Akaike Inf. Crit.           -1,512.913
=============================================
Note:              *p<0.1; **p<0.05; ***p<0.01
```

In [46]:

```r
# Load the stargazer package, which produces nicely formatted model output
library(stargazer)

# Estimate an MA(1) model on the growth rate series 'dy'
# - order = c(0,0,1) specifies MA(1), no differencing, no AR component
#   i.e. dy_t = intercept + alpha_1 * error_{t-1} + error_t
model.ma1 = arima(dy, order = c(0,0,1))

# Print the MA(1) estimation results in plain text format
# stargazer provides a cleaner regression-like summary than the default arima() output
stargazer(model.ma1, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                               dy
---------------------------------------------
ma1                           0.025
                             (0.060)

intercept                   0.007***
                             (0.001)

---------------------------------------------
Observations                   256
Log Likelihood               758.543
sigma2                        0.0002
Akaike Inf. Crit.           -1,511.087
=============================================
Note:              *p<0.1; **p<0.05; ***p<0.01
```

In [45]:

```r
# Load the stargazer package, which produces nicely formatted model output
library(stargazer)

# Estimate an ARMA(1,1) model on the growth rate series 'dy'
# - order = c(1,0,1) specifies AR(1) and MA(1), no differencing
#   i.e. dy_t = intercept + alpha_1 * dy_{t-1} + theta_1 * error_{t-1} + error_t
model.arma11 = arima(dy, order = c(1,0,1))

# Print the ARMA(1,1) estimation results in plain text format
# stargazer provides a cleaner regression-like summary than the default arima() output
stargazer(model.arma11, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                               dy
---------------------------------------------
ar1                         0.990***
                             (0.015)

ma1                         -0.964***
                             (0.024)

intercept                   0.008***
                             (0.002)

---------------------------------------------
Observations                   256
Log Likelihood               760.936
sigma2                        0.0002
Akaike Inf. Crit.           -1,513.873
=============================================
Note:              *p<0.1; **p<0.05; ***p<0.01
```

In [54]:

```r
# Collect AIC values into a data frame for comparison
# Note: lower AIC indicates better fit, but values are only strictly comparable
# if the models are estimated on the same number of observations
```

```r
AICdf = data.frame(
  ARMA00 = AIC(model.arma00),
  AR1    = AIC(model.ar1),
  MA1    = AIC(model.ma1),
  ARMA11 = AIC(model.arma11)
)

# Collect BIC values into a data frame for comparison
# Same caveat: direct comparison is not ideal when effective sample sizes differ
BICdf = data.frame(
  ARMA00 = BIC(model.arma00),
  AR1    = BIC(model.ar1),
  MA1    = BIC(model.ma1),
  ARMA11 = BIC(model.arma11)
)

# Display AIC and BIC tables
AICdf
BICdf
```

A data.frame: 1 × 4

ARMA00	AR1	MA1	ARMA11
<dbl>	<dbl>	<dbl>	<dbl>
-1512.913	-1511.1	-1511.087	-1513.873

A data.frame: 1 × 4

ARMA00	AR1	MA1	ARMA11
<dbl>	<dbl>	<dbl>	<dbl>
-1505.822	-1500.464	-1500.451	-1499.692

Model Selection (AIC and BIC)

We compared ARMA models using **AIC** (first table) and **BIC** (second table):

- **ARMA(0,0):**
  AIC = −1512.913, BIC = −1505.822
  → Baseline white noise model.
- **AR(1):**
  AIC = −1511.100, BIC = −1500.464
  → Worse fit than ARMA(0,0).
- **MA(1):**
  AIC = −1511.087, BIC = −1500.451
  → Worse fit than ARMA(0,0).
- **ARMA(1,1):**
  AIC = −1513.873 (**lowest AIC**) → best by AIC.
  BIC = −1499.692 (**highest BIC**) → worst by BIC.

Conclusion

- **AIC** suggests **ARMA(1,1)** is the best-fitting model.
- **BIC** suggests **ARMA(0,0)** (white noise) is the most appropriate.

Since BIC penalizes complexity more strongly, and the ACF/PACF showed no memory, the evidence leans toward **white noise**: the series has no significant autoregressive or moving-average structure.

GDP Growth and White Noise

Short-Run Dynamics

- The growth series may not have a predictable autoregressive (AR) or moving average (MA) structure.
- Past growth does not help predict future growth → consistent with the **random walk hypothesis** of GDP levels (so changes look unpredictable).

Modeling Implications

- At the **univariate time series** level, plain ARMA-type models may not add much value.
- But you can still:
  - Model **volatility (ARCH/GARCH)**, since growth rates often show time-varying variance.
  - Include **exogenous variables** (ARMAX, VAR, structural models) — GDP growth may depend on interest rates, shocks, or policy variables.
  - Look at **long-run levels** (GDP in logs often shows cointegration with other variables, even if growth rates look like white noise).

Economic Interpretation

- White-noise-like behavior of growth is consistent with the idea that **news and shocks drive growth** rather than past growth itself.
- That's why **forecasting GDP growth is hard** — surprises (policy, technology, global shocks) dominate.

Step 9

Export results

R: Broom function

In [56]:
```
library(broom)
write.csv(tidy(model.arma00), "output-arma00.csv")
```

R: Stargazer function

https://cran.r-project.org/web/packages/stargazer/vignettes/stargazer.pdf

In [57]:
```
library(stargazer)
stargazer(model.arma00, type = "text", out="output-arma00.doc")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                              dy
---------------------------------------------
intercept                  0.007***
                            (0.001)

---------------------------------------------
Observations                  256
Log Likelihood              758.456
sigma2                      0.0002
Akaike Inf. Crit.         -1,512.913
=============================================
Note:             *p<0.1; **p<0.05; ***p<0.01
```

Exercise 2

Find the best ARMA model for Personal Consumption Expenditure. https://fred.stlouisfed.org/series/PCE

Solution to Exercise 2

In [81]:
```
# Read the CSV file "PCE.csv" into a data frame called pce
pce = read.csv("PCE.csv")

# Display the first 6 rows of the dataset to check its structure
head(pce)

# Display the last 6 rows of the dataset to see the most recent observations
tail(pce)

# Convert the second column of pce into a time series object
# frequency = 12 means monthly data
# start = c(1959, 1) sets the beginning of the series at January 1959
tspce = ts(pce[, 2], frequency = 12, start = c(1959, 1))
```

A data.frame: 6 × 2

	observation_date	PCE
	<chr>	<dbl>
1	1959-01-01	306.1
2	1959-02-01	309.6
3	1959-03-01	312.7
4	1959-04-01	312.2
5	1959-05-01	316.1
6	1959-06-01	318.2

A data.frame: 6 × 2

	observation_date	PCE
	<chr>	<dbl>
794	2025-02-01	20436.3
795	2025-03-01	20578.5
796	2025-04-01	20621.1
797	2025-05-01	20617.5
798	2025-06-01	20693.1
799	2025-07-01	20802.0

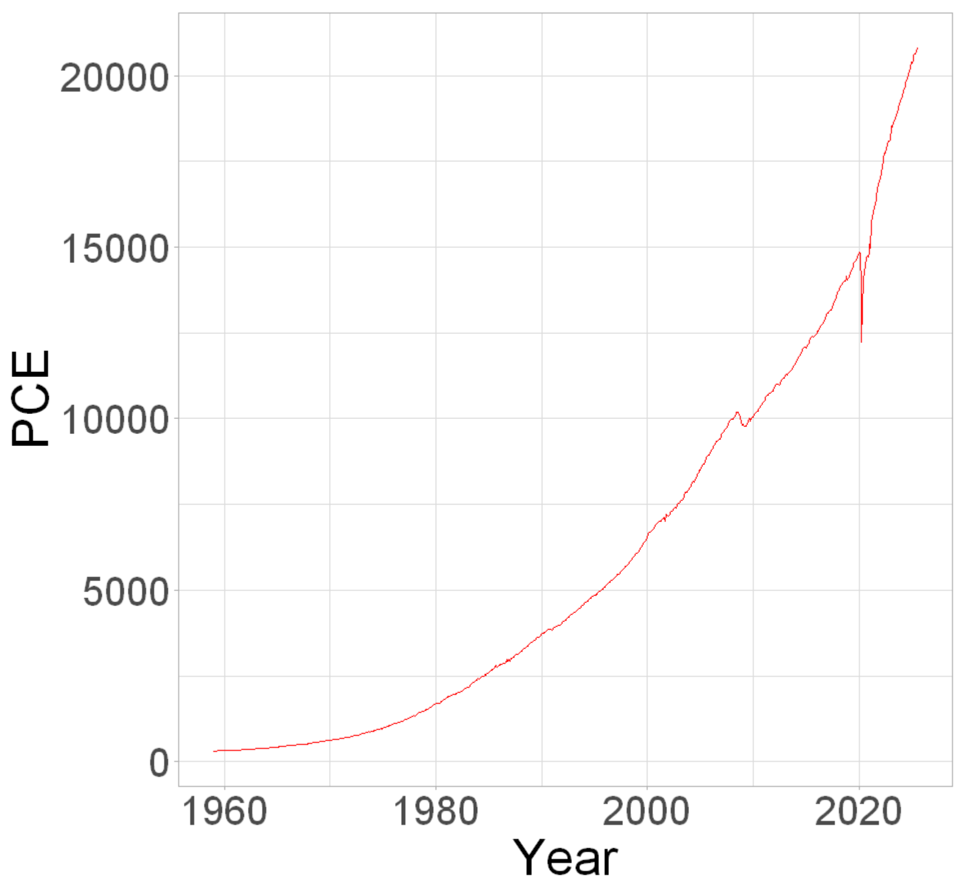
In [83]:

```
# Calculate the number of observations in the time series object
nd = length(tspce)

# Print a message along with the number of observations
cat("There are", nd, "observations in our data set.")
```

```
There are 799 observations in our data set.
```

In [84]:

```
# Load required libraries
library(ggplot2)   # for plotting
library(ggfortify) # for autoplot of time series objects

# Set default figure size (width = 8, height = 8)
options(repr.plot.width = 8, repr.plot.height = 8)

# Create an initial time series plot of tspce in red
fig = autoplot(tspce, colour = 'red')

# Customize the plot
fig = fig +
  theme(aspect.ratio = 1) +                        # make plot square-shaped
  theme_light() +                                  # use a light theme
  theme(aspect.ratio = 1) +                        # reinforce square aspect ratio
  theme(plot.margin = ggplot2::margin(0.2, 0.2, 0.2, 0.2, "cm")) + # adjust margins
  theme(text = element_text(size = 30)) +          # set large font size for readability
  labs(x = "Year") +                               # label x-axis
  labs(y = "PCE")                                  # label y-axis

# Display the plot
fig
```



In [85]:

```
library(urca)
summary(ur.df(tspce, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
     Min       1Q   Median       3Q      Max 
-1426.74    -9.39    -0.80    11.93   969.26 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  4.272945   8.531733   0.501  0.61663    
z.lag.1      0.004943   0.001867   2.648  0.00826 ** 
tt          -0.007376   0.043408  -0.170  0.86511    
z.diff.lag1  0.209692   0.035748   5.866 6.59e-09 ***
z.diff.lag2 -0.319135   0.036523  -8.738  < 2e-16 ***
z.diff.lag3  0.018993   0.036491   0.520  0.60287    
z.diff.lag4 -0.102876   0.035760  -2.877  0.00413 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 86.83 on 783 degrees of freedom
Multiple R-squared:  0.1606,	Adjusted R-squared:  0.1542 
F-statistic: 24.97 on 6 and 783 DF,  p-value: < 2.2e-16


Value of test-statistic is: 2.6478 34.6266 26.5819 

Critical values for test statistics: 
      1pct  5pct 10pct
tau3 -3.96 -3.41 -3.12
phi2  6.09  4.68  4.03
phi3  8.27  6.25  5.34
```

Interpretation of the Unit Root Test

Test statistics (type = "trend")

- **τ_3 = 2.65** vs **crit (1% = −3.96, 5% = −3.41, 10% = −3.12)** → **Fail to reject H_0** (unit root).

- **ϕ_2** and **ϕ_3** statistics far exceed critical values → **deterministic components (trend/drift) are significant**.

Conclusion

Log PCE is **non-stationary** with **one unit root** and a **deterministic trend**.

The series is **integrated of order one, I(1)**.

In [109…

```
# Take the first difference of the log of the PCE time series:
# - log(tspce) transforms PCE into logarithms (stabilizes variance, interpretable as growth)
# - diff(...) computes first differences (Δ log PCE)
# The result is an approximation of the monthly growth rate of PCE
temp = diff(log(tspce))
```

In [87]:

```
library(urca)
summary(ur.df(temp, type='trend', lags=8, selectlags="AIC"))
```

```
###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression trend


Call:
lm(formula = z.diff ~ z.lag.1 + 1 + tt + z.diff.lag)

Residuals:
      Min        1Q    Median        3Q       Max
-0.120628 -0.002771 -0.000043  0.003035  0.072161

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.858e-03  6.839e-04  11.489  < 2e-16 ***
z.lag.1     -1.097e+00  4.831e-02 -22.709  < 2e-16 ***
tt          -5.125e-06  1.296e-06  -3.956 8.33e-05 ***
z.diff.lag   1.636e-01  3.516e-02   4.653 3.83e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.008167 on 785 degrees of freedom
Multiple R-squared:  0.4861,    Adjusted R-squared:  0.4841
F-statistic: 247.5 on 3 and 785 DF,  p-value: < 2.2e-16


Value of test-statistic is: -22.7092 171.9038 257.8553

Critical values for test statistics:
      1pct  5pct 10pct
tau3 -3.96 -3.41 -3.12
phi2  6.09  4.68  4.03
phi3  8.27  6.25  5.34
```

Testing for the Unit Root

Test statistics (type = "trend")
- τ_3 **= −22.7** → far below critical values → **strongly reject H_0** (unit root).
- ϕ_2 and ϕ_3 again exceed critical values → **deterministic trend is present** in the differenced series.

Conclusion

Growth rates are **trend-stationary**:
- The stochastic trend (unit root) has been removed by differencing.
- But the mean of Δ log PCE still **depends on time (t)**, so it has a deterministic trend.
- To obtain a stationary series with constant mean, we would need to **detrend** (regress on time and use residuals) or include a **trend term** in further modeling.

Overall Takeaway

- **Levels:** I(1), with deterministic trend.
- **Growth rates:** I(0), but still trend-stationary.
- For modeling, either **detrend explicitly** or **include a deterministic trend** in the regression.

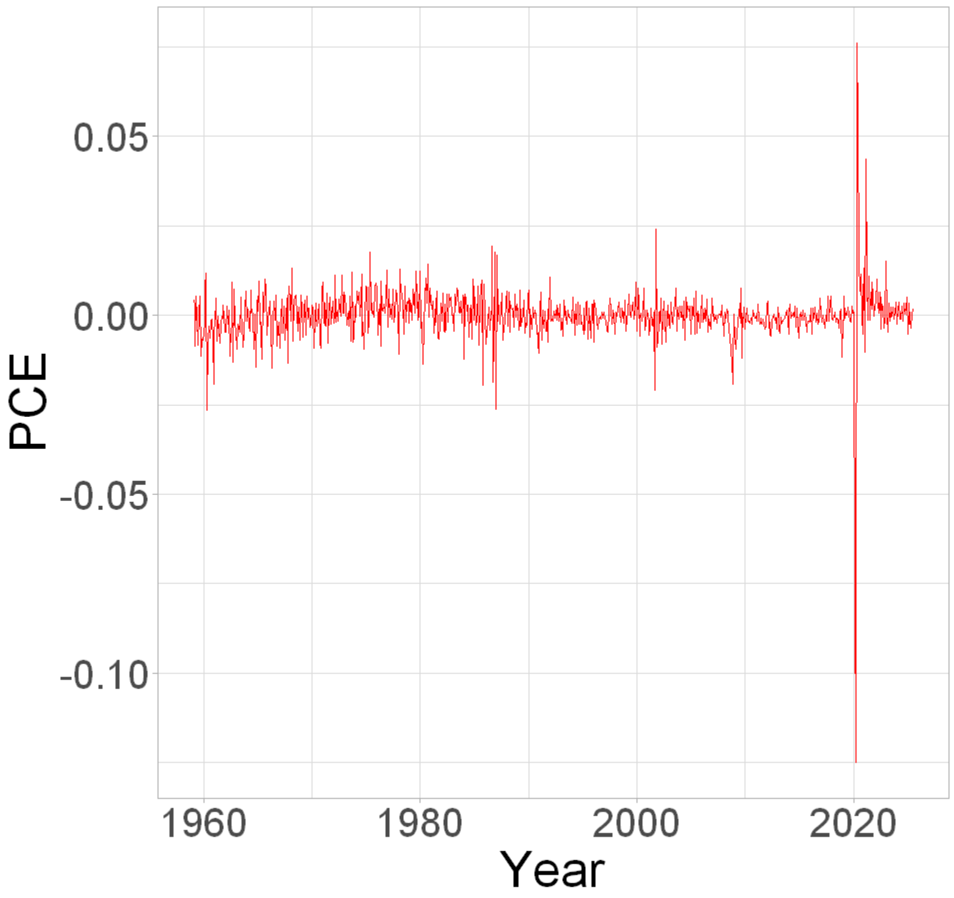
In [112…

```
# Load the pracma package (provides the detrend() function)
library(pracma)

# Remove the linear trend from the series "temp"
# as.numeric(temp) ensures pracma::detrend() works
# (expects a numeric vector, not a ts object)
# 'linear' specifies linear detrending
temp = detrend(as.numeric(temp), 'linear')

# Define the start date for the time series
dpce = ts(temp, frequency = 12, start = c(1959, 2))
```

In [113…

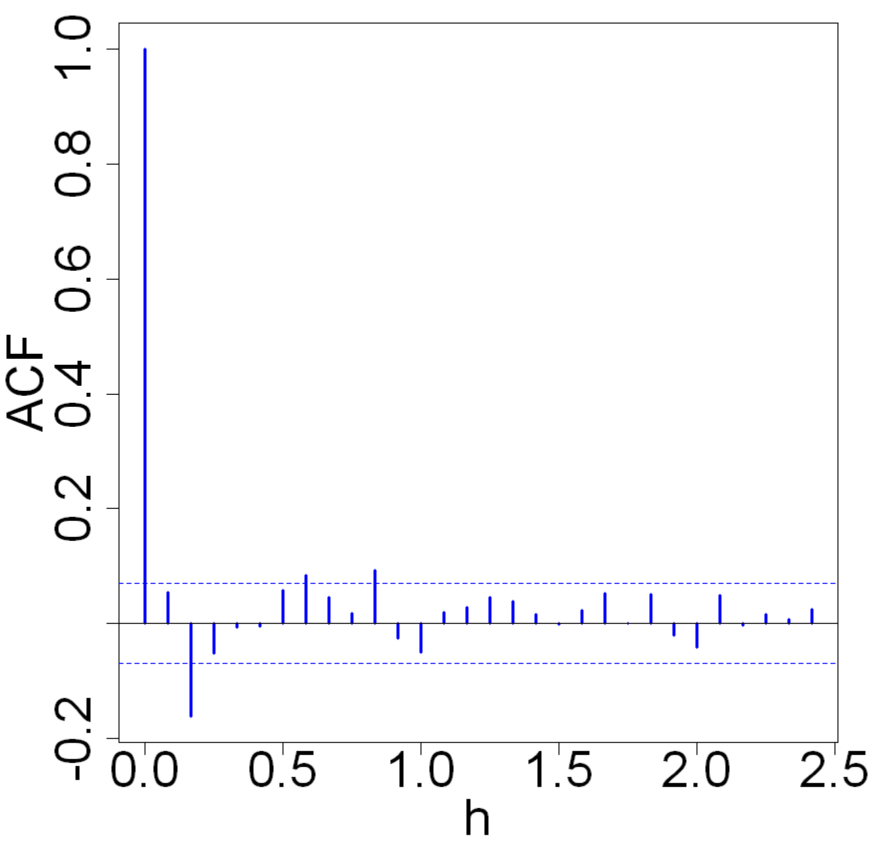
```
# Load required libraries
library(ggplot2)   # for plotting
library(ggfortify) # for autoplot of time series objects

# Set default figure size (width = 8, height = 8)
options(repr.plot.width = 8, repr.plot.height = 8)

# Create an initial time series plot of tspce in red
fig = autoplot(dpce, colour = 'red')

# Customize the plot
fig = fig +
  theme(aspect.ratio = 1) +                        # make plot square-shaped
  theme_light() +                                  # use a light theme
  theme(aspect.ratio = 1) +                        # reinforce square aspect ratio
  theme(plot.margin = ggplot2::margin(0.2, 0.2, 0.2, 0.2, "cm")) + # adjust margins
```

```
  theme(text = element_text(size = 30)) +          # set large font size for readability
  labs(x = "Year") +                               # label x-axis
  labs(y = "PCE")                                  # label y-axis

# Display the plot
fig
```



In [114…

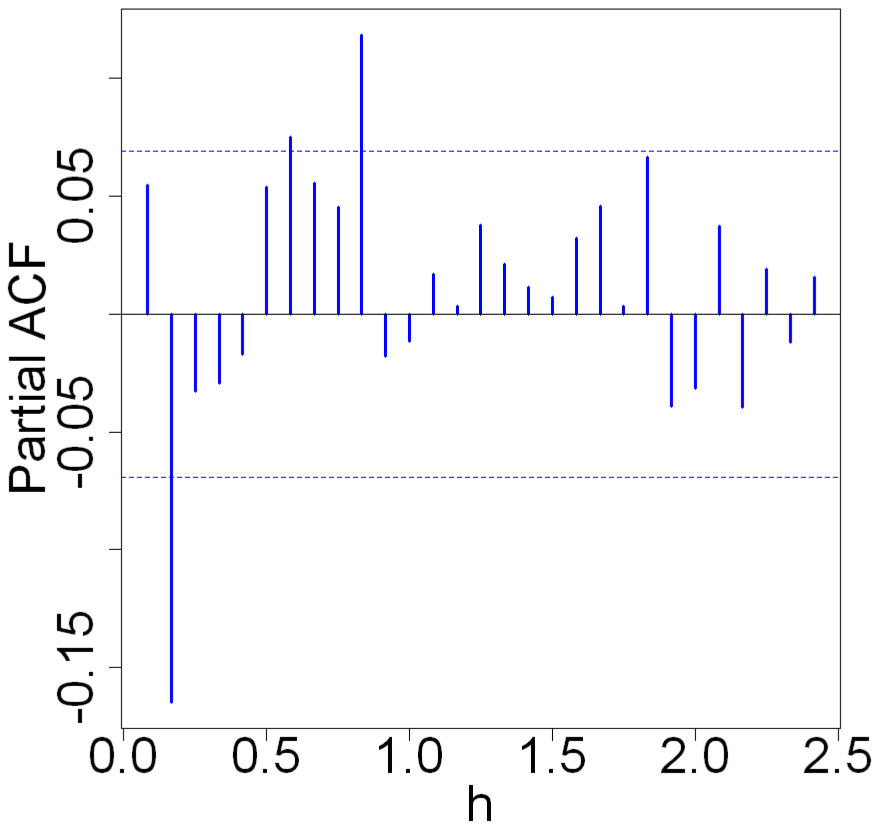
```
# Compute the autocorrelation function (ACF) of dy without plotting
acfun = acf(dpce, plot = FALSE)

# Set the size of the plotting window (useful in Jupyter / R notebooks)
options(repr.plot.width = 8, repr.plot.height = 8)

# Adjust margins: bottom, left, top, right
par(mar = c(5, 5, 5, 5))

# Plot the ACF with custom styling
plot(acfun,
     col = "blue",          # bars in blue
     main = "",             # remove the default main title
     cex = 2.5,             # scale size of points (not critical here)
     cex.lab = 2.5,         # enlarge axis labels (xlab, ylab)
     cex.axis = 2.5,        # enlarge tick labels
     xlab = "h",            # label for x-axis (lag h)
     lwd = 3)               # line width for bars
```



In [115…

```
# Compute the autocorrelation function (ACF) of dy without plotting
pacfun = pacf(dpce, plot = FALSE)

# Set the size of the plotting window (useful in Jupyter / R notebooks)
options(repr.plot.width = 8, repr.plot.height = 8)
```

```r
# Adjust margins: bottom, left, top, right
par(mar = c(5, 5, 5, 5))

# Plot the ACF with custom styling
plot(pacfun,
     col = "blue",          # bars in blue
     main = "",             # remove the default main title
     cex = 2.5,             # scale size of points (not critical here)
     cex.lab = 2.5,         # enlarge axis labels (xlab, ylab)
     cex.axis = 2.5,        # enlarge tick labels
     xlab = "h",            # label for x-axis (lag h)
     lwd = 3)               # line width for bars
```



Interpretation of the ACF/PACF Plots

Textbook Rules

AR(p) - **PACF:** cutoff at lag *p*.
- **ACF:** decays gradually (often geometrically or with damped oscillations). **MA(q)** - **ACF:** cutoff at lag *q*.
- **PACF:** decays gradually. **ARMA(p, q)** - **ACF & PACF:** both decay gradually (no sharp cutoffs).

What Our Plots Show

1) ACF (Autocorrelation Function)
- Lag 0 = 1 (by construction).
- **Lag 1:** very small / insignificant.
- **Lag 2:** negative and clearly outside confidence bands (significant).
- After lag 2: values hover near zero; no systematic pattern. → The **first clear significant autocorrelation** is at **lag 2**. **2) PACF (Partial Autocorrelation Function)** - No single clean cutoff.
- Largest negative spike around **lag 2**, with other weak scattered spikes.

Implications for Model Choice

- **AR(1):** would show a slowly decaying ACF and a sharp PACF cutoff at lag 1 → **Not observed**.
- **MA(1):** would show an ACF cutoff at lag 1 and a gradually decaying PACF → **Not observed**.
- **AR(2):** commonly produces an ACF with a notable spike at lag 2 and a PACF that cuts off at lag 2 (or is dominated by lag-2 effects) → **Most consistent with our plots**.

Conclusion

Start with **AR(2)** as the leading candidate (`ARIMA(2,0,0)` on the detrended/differenced series). Then compare against nearby alternatives (e.g., `ARMA(2,1)`) using **AIC/BIC** and residual diagnostics (e.g., **Ljung–Box**) to confirm.

In [121…

```r
# Load the stargazer package (for nicely formatted model output)
library(stargazer)

# Estimate an AR(2) model on the detrended PCE series (dpce)
# order = c(2,0,0) means:
#   2 autoregressive lags (AR terms),
#   0 differencing,
#   0 moving-average terms
model.ar2 = arima(dpce, order = c(2,0,0))
```

```r
# Display the AR(2) model results in text format
# stargazer works best with lm/glm objects, but it will print
# the arima object too (coefficients, standard errors, etc.)
stargazer(model.ar2, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                             dpce
---------------------------------------------
ar1                         0.064*
                           (0.035)

ar2                        -0.164***
                           (0.035)

intercept                  -0.00000
                           (0.0003)

---------------------------------------------
Observations                 798
Log Likelihood            2,708.134
sigma2                      0.0001
Akaike Inf. Crit.        -5,408.268
=============================================
Note:             *p<0.1; **p<0.05; ***p<0.01
```

In [123…

```r
# Load stargazer (if not already loaded)
library(stargazer)

# Estimate an AR(1) model on the detrended PCE series (dpce)
# order = c(1,0,0) means:
#   1 autoregressive lag,
#   0 differencing,
#   0 moving-average terms
model.ar1 = arima(dpce, order = c(1,0,0))

# Display the AR(1) model results in text format
stargazer(model.ar1, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                             dpce
---------------------------------------------
ar1                          0.055
                           (0.035)

intercept                  -0.00000
                           (0.0003)

---------------------------------------------
Observations                 798
Log Likelihood            2,697.168
sigma2                      0.0001
Akaike Inf. Crit.        -5,388.335
=============================================
Note:             *p<0.1; **p<0.05; ***p<0.01
```

In [126…

```r
# Load stargazer (if not already loaded)
library(stargazer)

# Estimate an MA(1) model on the detrended PCE series (dpce)
# order = c(0,0,1) means:
#   0 autoregressive lags,
#   0 differencing,
#   1 moving-average term
model.ma1 = arima(dpce, order = c(0,0,1))

# Display the MA(1) model results in text format
stargazer(model.ma1, type = "text")
```

```
=============================================
                      Dependent variable:
                  ---------------------------
                             dpce
---------------------------------------------
ma1                         0.080*
                           (0.042)

intercept                   0.00000
                           (0.0003)

---------------------------------------------
Observations                 798
Log Likelihood            2,697.733
sigma2                      0.0001
Akaike Inf. Crit.        -5,389.465
=============================================
Note:             *p<0.1; **p<0.05; ***p<0.01
```

In [125…

```r
# Collect and compare model fit statistics (AIC and BIC) across models

# Create a data frame with the Akaike Information Criterion (AIC) values
```

```r
# for the AR(2), AR(1), and MA(1) models
AICdf = data.frame(
  AR2 = AIC(model.ar2),   # AIC of AR(2)
  AR1 = AIC(model.ar1),   # AIC of AR(1)
  MA1 = AIC(model.ma1)    # AIC of MA(1)
)

# Create a data frame with the Bayesian Information Criterion (BIC) values
# for the same three models
BICdf = data.frame(
  AR2 = BIC(model.ar2),   # BIC of AR(2)
  AR1 = BIC(model.ar1),   # BIC of AR(1)
  MA1 = BIC(model.ma1)    # BIC of MA(1)
)

# Print the AIC comparison table
AICdf

# Print the BIC comparison table
BICdf
```

A data.frame: 1 × 3

AIC.model.ar2.	AIC.model.ar1.	AIC.model.ma1.
<dbl>	<dbl>	<dbl>
-5408.268	-5388.335	-5389.465

A data.frame: 1 × 3

BIC.model.ar2.	BIC.model.ar1.	BIC.model.ma1.
<dbl>	<dbl>	<dbl>
-5389.54	-5374.289	-5375.419

Model Comparison: AIC and BIC Results

AIC Results: - AR(2) = −5408.27
- AR(1) = −5388.34
- MA(1) = −5389.47 **BIC Results:**
- AR(2) = −5389.54
- AR(1) = −5374.29
- MA(1) = −5375.42

Interpretation

- Both **AIC** and **BIC** are lowest (most negative) for the **AR(2)** model.
- This confirms what we suspected from the ACF and PACF: the **AR(2)** specification fits the data better than AR(1) or MA(1).
- The improvement over AR(1) and MA(1) is substantial enough to justify the additional parameter.

Conclusion

The **AR(2)** model is the preferred specification among the candidates tested.

In [ ]: