# Lecture 1, September 3, 2025

# Stationary series

Marta Boczon

Department of Economics

Copnehagen Business School

mbo.eco@cbs.dk

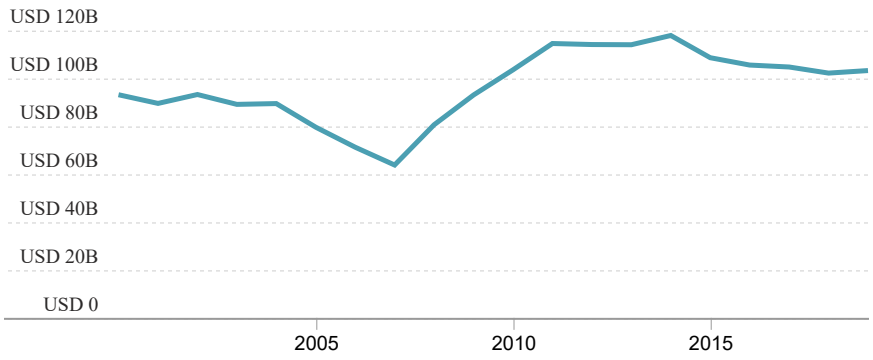
# Introduction

1. In contrast to cross sectional data, time series econometrics concerns itself with the evolvement of variables over time.
2. Enders says: "The task facing the modern time-series econometrician is to develop reasonably simple models capable of forecasting, interpreting, and testing hypotheses concerning economic data."
3. A time series is a sequential set of data points, measured typically over successive times.
4. Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data.
5. A time series containing records of a single variable is termed as univariate, but if records of more than one variable are considered then it is termed as multivariate.
6. In general, a time series is affected by four components, i.e., trend, seasonal,cyclical and irregular components.

Examples of time series data:

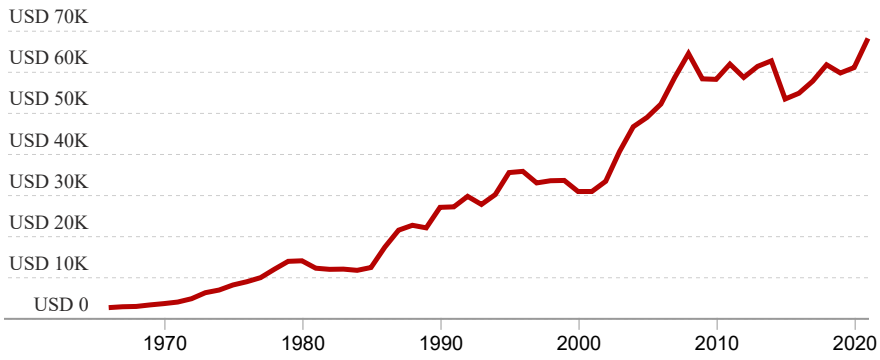
1. Governent debt in Denmark
2. Gross domestic product per capita in Denmark
3. Labor force participation rate in Denmark
4. Market capitalization of domestic companies in Denmark
5. Unemployment rate in Denmark
6. COVID-19 cumulative deaths in Denmark

## Government debt in Denmark



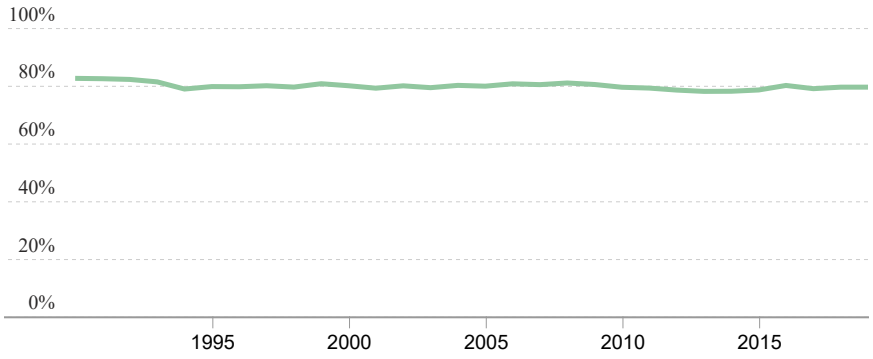

Data from ec.europa.eu via Data Commons

## Gross domestic product per capita in Denmark

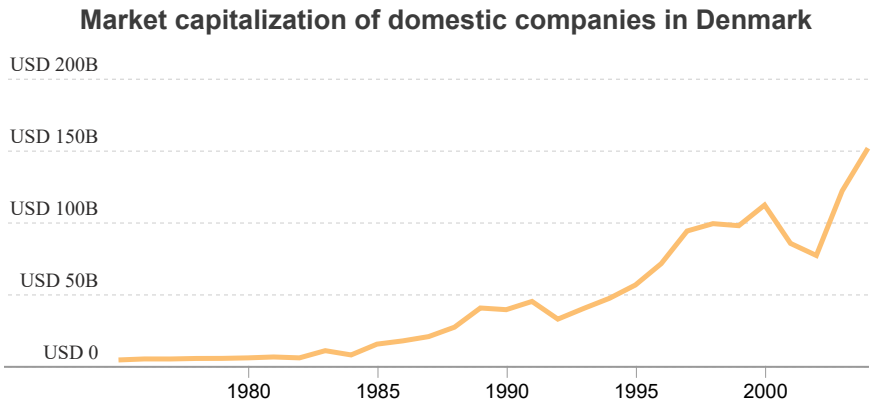



Data from datacatalog.worldbank.org via Data Commons

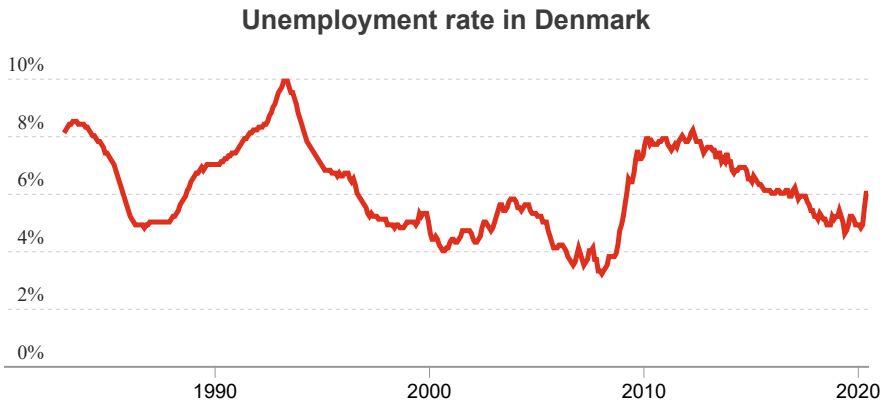
## Labor force participation rate in Denmark



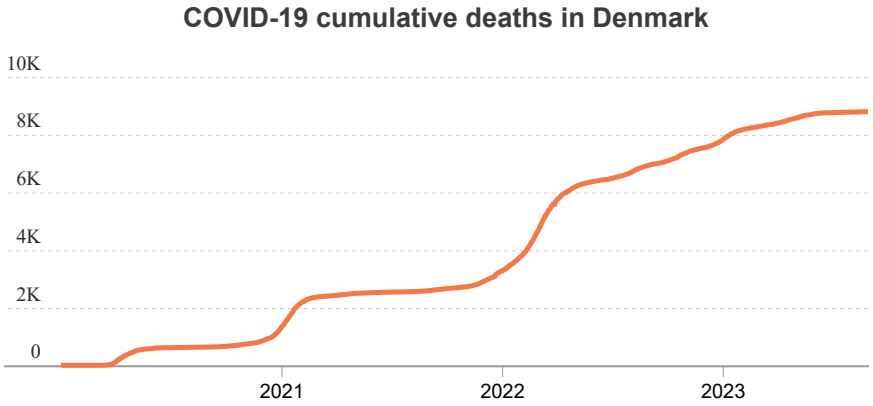

Data from datacatalog.worldbank.org via Data Commons




Data from datacatalog.worldbank.org via Data Commons




Data from ec.europa.eu via Data Commons

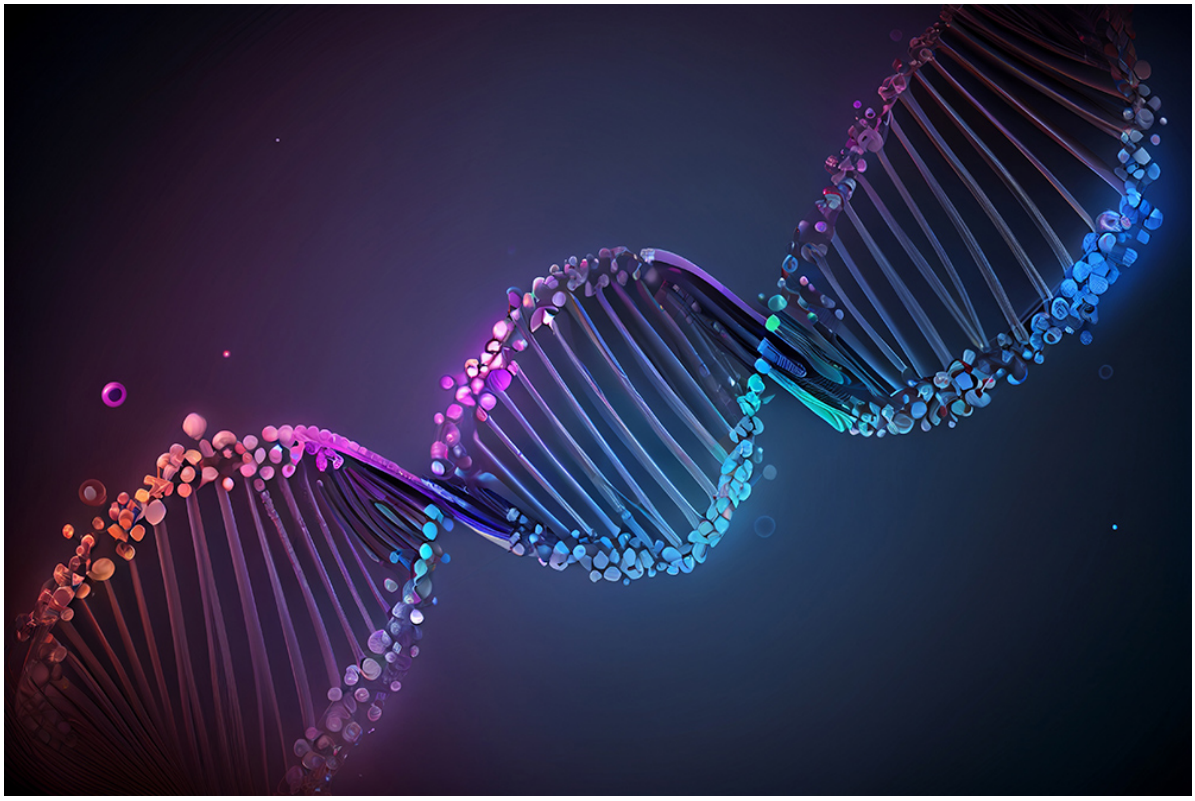



Data from who.int via Data Commons

Examples of sources of data series data:

1. Eurostat
2. FRED

# Building up an intuition: Time series vs DNA analysis



Just as a time series is the observed realization of an underlying data-generating process (DGP), DNA can be thought of as the biological DGP that generates the traits we see in humans. In time series, the DGP encodes the rules and dependencies—like how today's value depends on past values—and the actual time series we observe is just one path that could have been produced under those rules. Similarly, DNA encodes the instructions that, together with environmental influences, generate observable characteristics such as eye color, height, or behavior. What we observe in an individual is just one realization of what the DNA "process" allows. In both cases, the underlying structure is invisible, but it dictates the patterns and variations in the realizations we study

# Introduction to an autoregressive process AR

Autoregressive process provides a half of the backbone time-series ARMA model. We can define an autoregressive sequence of order $p$, denoted by AR($p$):

$$y_t = \sum_{i=1}^{p} \beta_i y_{t-i} + \epsilon_t,$$

where $y_t$ is the value of the time series at time $t$, $\beta_i$ are the autoregressive coefficients that capture how past values influence the present, and $\epsilon_t$ is the error term (or innovation) at time $t$.

As we can see the model is a linear function of the previous $y$ as well as of the current shock (with coefficient 1). Intuitively, an AR($p$) process models the present as a weighted sum of the $p$ most recent past values plus a new shock.

**Examples:**

AR(1):

$$y_t = \beta_1 y_{t-1} + \epsilon_t$$

AR(2):

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t$$

AR(3):

$$y_t = \beta_1 y_{t-1} + \beta_2 y_{t-2} + \beta_3 y_{t-3} + \epsilon_t$$

**Estimation:** This model can be estimated via OLS or maximum likelihood among others because the predictors are directly observed.

# R: Generating an AR(1) process

In [1]:
```r
# Define the number of time periods (observations) to simulate
T = 200
T
```

200

In [2]:
```r
# Initialize a vector of length T with zeros to store the AR(1) process values
ar1 = rep(0, T)

# Show the first five entires of the vector ar1
head(ar1)

# Show the last five entires of the vector ar1
tail(ar1)
```

0 · 0 · 0 · 0 · 0 · 0

0 · 0 · 0 · 0 · 0 · 0

In [3]:
```r
# Set the coefficient (AR(1) parameter)
beta = 0.8
beta
```

0.8

In [4]:
```r
# Generate random noise (innovations) from a Normal distributio
# rnorm(n, mean = 0, sd = 1) generates 'n' draws from N(mean, sd)
# Set seed for reproducibility
set.seed(123)
e = rnorm(T, 0, 0.2)
```

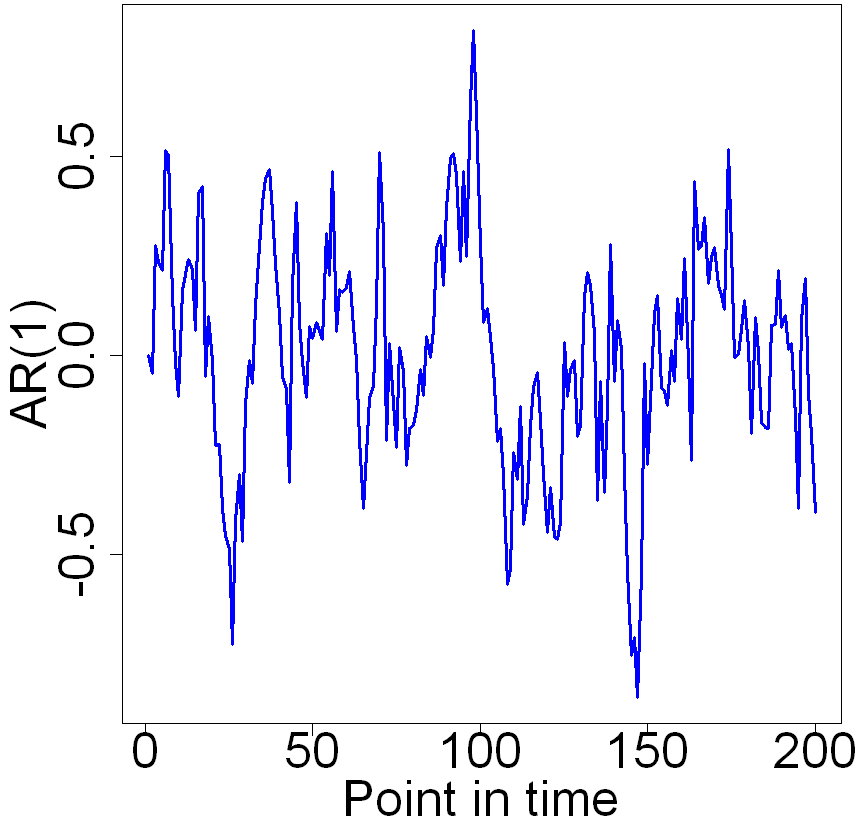
```r
# Display the first six shocks
head(e)
```

-0.112095129310443 · -0.046035497896656 · 0.311741662829825 · 0.0141016782849152 · 0.0258575470321892 · 0.343012997376656

In [5]:
```r
# Generate the AR(1) process recursively
for (t in 2:T) {
    ar1[t] = beta * ar1[t-1] + e[t]
    # Each value depends on the previous one plus a new random shock
    # Start at t = 2 because ar1[t] depends on ar1[t-1];
    # there is no ar1[0], so ar1[1] must be set in advance (here initialized to 0)
}

# Show the first six simulated values of the AR(1) process
head(ar1)
```

0 · -0.046035497896656 · 0.2749132645125 · 0.234032289894915 · 0.213083378948121 · 0.513479700535153

In [6]:
```r
# Set plot dimensions for Jupyter/IRkernel (optional, ignored in base R)
options(repr.plot.width=8, repr.plot.height=8)

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

# Plot the simulated AR(1) process as a line graph
plot(
  ar1,
  type = "l",              # line plot
  col = "blue",            # line color
  lwd = 3,                 # line width
  xlab = "Point in time",  # x-axis label
  ylab = "AR(1)",          # y-axis label
  cex.lab = 2.5,           # scaling factor for axis labels
  cex.axis = 2.5           # scaling factor for axis tick labels
)
```




# Introduction to a moving average process MA

The moving-average model provides the other half of the backbone time-series ARMA model. We can define a moving average sequence of order $q$, denoted by MA($q$):

$$y_t = \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t,$$

where $y_t$ is the value of the process at time $t$, $\beta_i$ are the coefficients, and $\varepsilon_t$ represents the shocks — unobservable random disturbances. These shocks are not measurable external events like weather, but rather the model's error terms: the unexpected part of $y_t$ that remains after accounting for everything predictable from the past.

**Examples:**

MA(1):

$$y_t = \beta_1 \epsilon_{t-1} + \epsilon_t$$

MA(2):

$$y_t = \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \epsilon_t$$

MA(3):

$$y_t = \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \beta_3 \epsilon_{t-3} + \epsilon_t$$

**Estimation:** This model cannot be estimated via OLS because the predictors are unobservable. However, it can be estimated using maximum likelihood.

# R: Generating MA(1) process

In [7]:
```r
# Define the number of time periods (observations) to simulate
T = 200
T
```

200

In [8]:
```r
# Initialize a vector of length T with zeros to store the MA(1) process values
ma1 = rep(0, T)

# Show the first five entires of the vector ma1
head(ma1)

# Show the last five entires of the vector ma1
tail(ma1)
```

0 · 0 · 0 · 0 · 0 · 0

0 · 0 · 0 · 0 · 0 · 0

In [9]:
```r
# Set the coefficient (MA(1) parameter)
beta = 0.8
beta
```

0.8

In [10]:
```r
# Generate random noise (innovations) from a Normal distributio
# rnorm(n, mean = 0, sd = 1) generates 'n' draws from N(mean, sd)
# Set seed for reproducibility
set.seed(123)
e = rnorm(T, 0, 0.2)

# Display the first six shocks
head(e)
```

-0.112095129310443 · -0.046035497896656 · 0.311741662829825 · 0.0141016782849152 · 0.0258575470321892 · 0.343012997376656

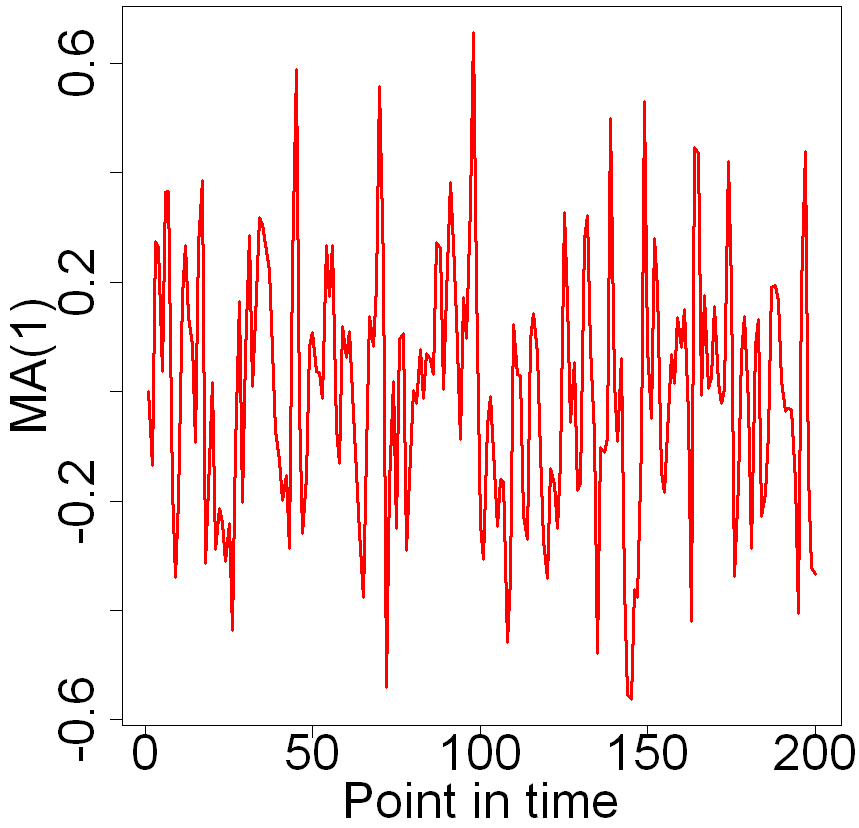
In [11]:
```r
# Generate the MA(1) process recursively
for (t in 2:T) {
    ma1[t] = beta * e[t-1] + e[t]
    # Each value depends on the current shock plus the previous shock (lagged error te
    # Start at t = 2 because we need e[t-1]; there is no e[0]
}
```

```r
# Show the first six simulated values of the MA(1) process
head(ma1)
```

0 · -0.13571160134501 · 0.2749132645125 · 0.263495008548775 · 0.0371388896601214 · 0.363699035002408

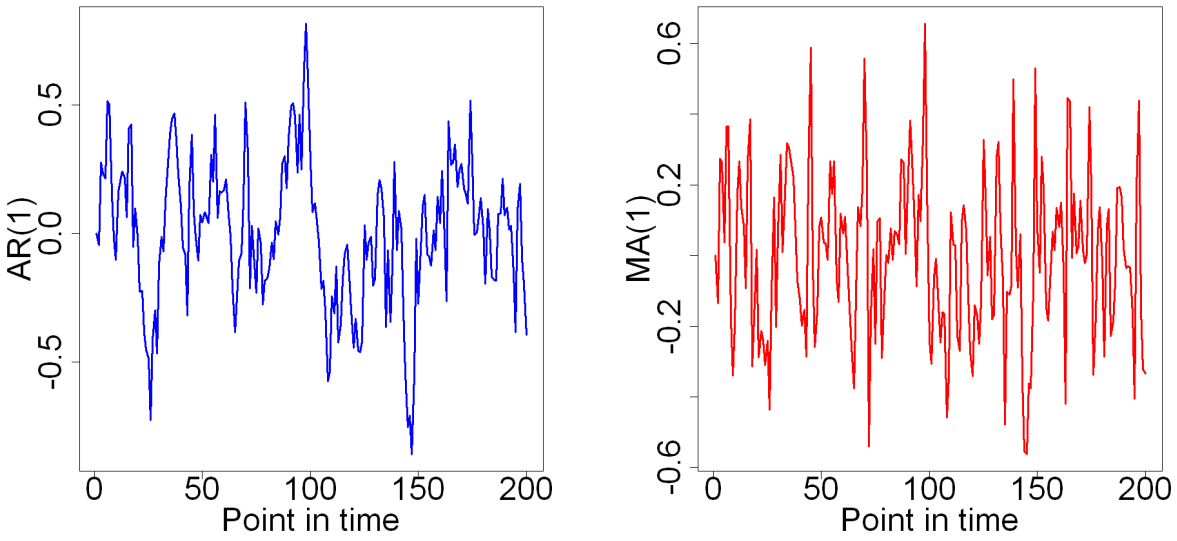
In [12]:

```r
# Set plot dimensions for Jupyter/IRkernel (optional, ignored in base R)
options(repr.plot.width=8, repr.plot.height=8)

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

# Plot the simulated MA(1) process as a line graph
plot(
  ma1,
  type = "l",              # line plot
  col = "red",             # line color
  lwd = 3,                 # line width
  xlab = "Point in time",  # x-axis label
  ylab = "MA(1)",          # y-axis label
  cex.lab = 2.5,           # scaling factor for axis labels
  cex.axis = 2.5           # scaling factor for axis tick labels
)
```




# Comparison of AR and MA processes

In [13]:
```r
# Set plot dimensions for Jupyter/IRkernel (ignored in base R)
options(repr.plot.width = 16, repr.plot.height = 8)

# Adjust plot margins (bottom, left, top, right) in lines of text
par(mar = c(5, 5, 5, 5))

# Arrange plots in a 1x2 grid (1 row, 2 columns)
par(mfrow = c(1, 2))

# Plot the simulated AR(1) process
plot(
  ar1,
  type = "l",                # line plot
  col = "blue",              # line color
  lwd = 3,                   # line width
  xlab = "Point in time",    # x-axis label
  ylab = "AR(1)",            # y-axis label
  cex.lab = 2.5,             # axis label size
  cex.axis = 2.5             # axis tick label size
)

# Plot the simulated MA(1) process
plot(
  ma1,
  type = "l",                # line plot
  col = "red",               # line color
  lwd = 3,                   # line width
  xlab = "Point in time",    # x-axis label
  ylab = "MA(1)",            # y-axis label
  cex.lab = 2.5,             # axis label size
  cex.axis = 2.5             # axis tick label size
)
```




# Exercise

Simulate 1,000 observations from an AR(3) process, and an MA(3) process, using the coefficients:

$\beta_1 = 0.2$

$\beta_2 = 0.1$

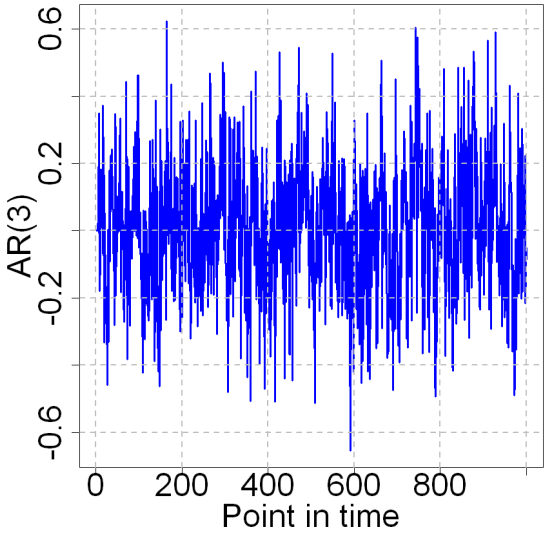
$\beta_3 = 0.2$

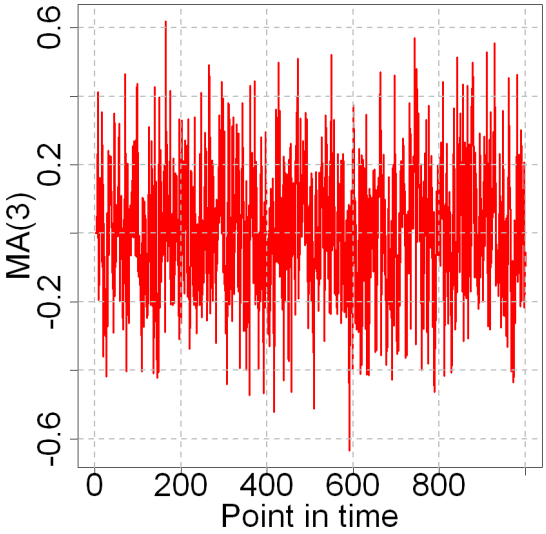
# Solution:

In [14]:

```r
# Define the number of time periods (observations) to simulate
T = 1000

# Initialize vectors with zeros to store the AR(3) and MA(3) process values
ar3 = rep(0, T) # AR(3) series
ma3 = rep(0, T) # MA(3) series

# Set the coefficients for the AR(3) and MA(3) models
beta1 = 0.2
beta2 = 0.1
beta3 = 0.2

# Generate shocks (innovations) from a Normal distribution
set.seed(123)               # set seed for reproducibility
e = rnorm(T, mean = 0, sd = 0.2)

# Generate the AR(3) process
for (t in 4:T) {
  # AR(1) recursion
  # Start at t = 4 because the AR(3) recursion needs the three previous values:
  # ar3[t-1], ar3[t-2], and ar3[t-3].
  # For t < 4, not all these lags exist.
  ar3[t] = beta1 * ar3[t-1] + beta2 * ar3[t-2] + beta3 * ar3[t-3] + e[t]
}

# Generate the MA(3) process
for (t in 4:T) {
  # MA(3) recursion
  # Start at t = 4 because the MA(3) recursion needs the three previous shocks:
  # e[t-1], e[t-2], and e[t-3].
  # For t < 4, not all these lags exist.
  ma3[t] = beta1 * e[t-1] + beta2 * e[t-2] + beta3 * e[t-3] + e[t]
}

# Set plot dimensions for Jupyter/IRkernel (ignored in base R)
options(repr.plot.width = 16, repr.plot.height = 8)

# Adjust plot margins (bottom, left, top, right) in lines of text
par(mar = c(5, 5, 5, 5))

# Arrange plots in a 1x2 grid (1 row, 2 columns)
par(mfrow = c(1, 2))

# Plot a zoomed-in window around the shock
plot(
  c(1:T), ar3,
  type = "l", col = "blue", lwd = 3,              # line plot
  xlab = "Point in time", ylab = "AR(3)",       # axis labels
```

```r
  cex.lab = 2.5, cex.axis = 2.5                  # scaling factors
)

# Add gridlines for readability
grid(nx = NULL, ny = NULL,
     lty = 2,      # dashed lines
     col = "gray", # grid color
     lwd = 2)      # grid width

# Plot a zoomed-in window around the shock
plot(
  c(1:T), ma3,
  type = "l", col = "red", lwd = 3,              # line plot
  xlab = "Point in time", ylab = "MA(3)",        # axis labels
  cex.lab = 2.5, cex.axis = 2.5                  # scaling factors
)

# Add gridlines for readability
grid(nx = NULL, ny = NULL,
     lty = 2,      # dashed lines
     col = "gray", # grid color
     lwd = 2)      # grid width
```




https://python.plainenglish.io/how-to-convert-google-colab-notebook-ipynb-to-html-ccfeda199246

%%shell

jupyter nbconvert --to html ///content/sample_data/lecture_1.ipynb