Financial Econometrics - Tutorials

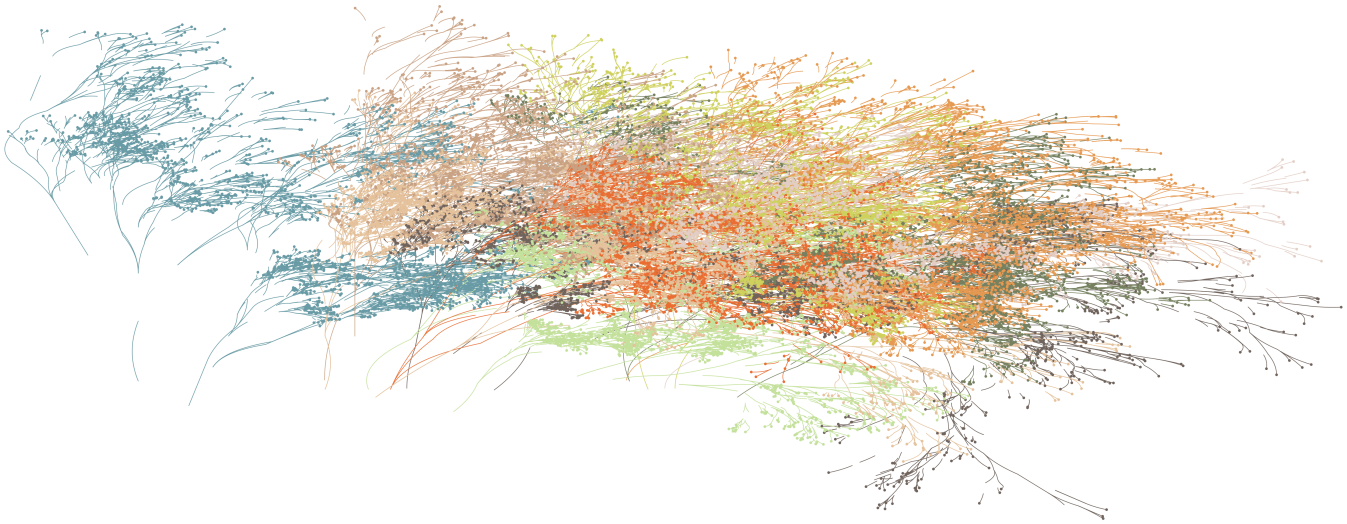
Alessandro Ciancetta

Last update: January 25, 2024

Contents

About

TA materials for the Financial Econometrics course held by Prof. Christian Brownlees at the Barcelona School of Economics.



Chapter 1

Introduction to time series

1.1 Stochastic processes and dependence

Stochastic processes are a tool for modeling dependence in consecutive random variables $\{\dots, Y_{-2}, Y_{-1}, Y_0, Y_1, Y_2 \dots\}$. However, in practice, when we observe an empirical time series we are considering *one, truncated* realization of the stochastic process, $\{y_1, y_2, \dots, y_T\}$. As it is easy to imagine, this can cause some issues in studying the properties of an empirical time series. First, because we can only study the *finite* dimensional distribution of the process. Second, because our task is to learn something about the process by knowing only one realization of it.

To overcome this limitations we need assumptions. In particular, two common assumptions in time series analysis are, loosely speaking:

- stationarity: the observed values in the sequence come from the same distribution, so that it is possible to learn from the past observations and to generalize the results to the entire, infinite stochastic process
- ergodicity: values observed far away in time can be considered as independent, and hence if we have enough observations the empirical time series is representative of the entire distribution of the stochastic process

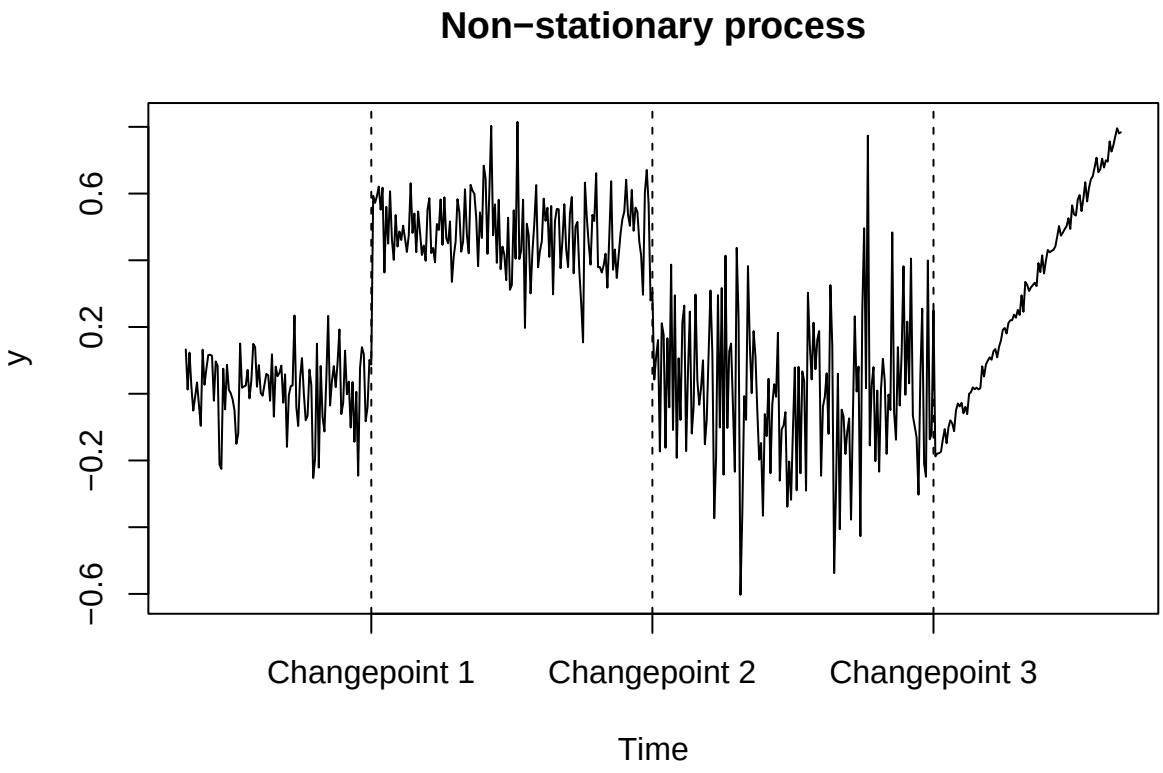
Under these (or similar) assumptions, we can use the observations from a single empirical time series to learn the parameters of our models.

Stationarity

Consider the example in the plot below. If we had only observations laying between two changepoints we would not be able to retrieve the dynamics of the underlying process. For instance, if we observed a series ending before the first changepoint, we would have no information about the future realizations of the process. Indeed, future observations would come from different distributions that we could not learn from available information.

```
## Example 1: non-stationary process
# simulation
t_max <- 500
y <- rep(NA, t_max)
for (t in 1:t_max) {
  if (t<=100) {
    y[t] <- rnorm(1, mean = 0, sd = 0.1)
  }
  if (t>100 & t<=250) {
    y[t] <- rnorm(1, mean = 0.5, sd = 0.1)
  }
  if (t>250 & t<=400) {
    y[t] <- rnorm(1, mean = 0, sd = 0.2)
  }
  if (t>400 & t<=t_max) {
```

```
    y[t] <- 0.01*(t-400) + rnorm(1, mean = -0.2, sd = 0.02)
  }
}
# plot
plot.ts(y, main = "Non-stationary process", xaxt = "n")
abline(v = c(100, 250, 400), lty = 2)
axis(1, at=c(100, 250, 400),
     labels = c("Changepoint 1", "Changepoint 2", "Changepoint 3"))
```



Ergodicity

Let $\{z_t\}_{t=1}^{T} \overset{iid}{\sim} \mathcal{N}(0,1)$. Consider the two following data-generating processes:

$$y_t = U_0 + 0.25 z_t, \quad \text{with } U_0 \sim \mathcal{N}(0, 100)$$
$$x_t = z_t + z_{t-1}$$

We have:

$$\mathbb{E}[y_t] = \mathbb{E}[x_t] = 0$$

$$\text{cov}(y_t, y_{t-k}) = \begin{cases} 100 + 0.25^2 & k = 0 \\ 100 & k \neq 0 \end{cases}$$

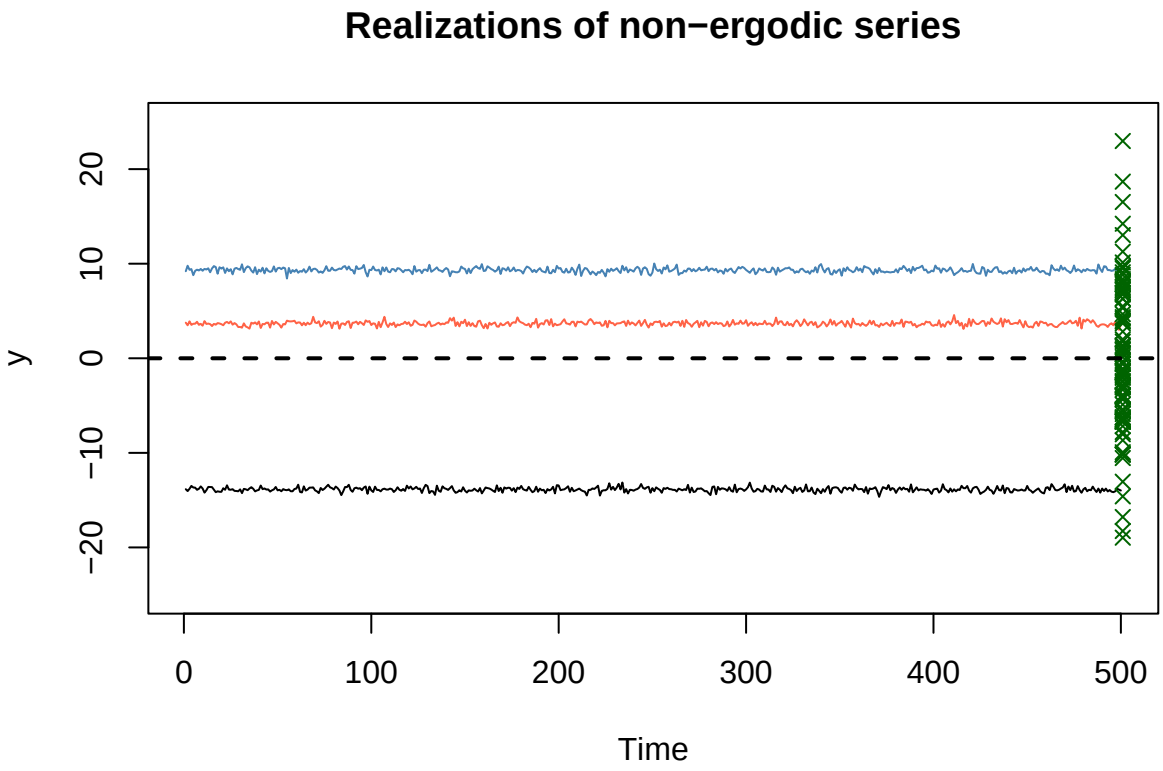
$$\text{cov}(x_t, x_{t-k}) = \begin{cases} 2 & k = 0 \\ 1 & k = 1 \\ 0 & k \geq 2 \end{cases}$$

We now simulate the two data generating processes in R. We want to draw many sequences $\{y_1, \dots, y_T\}^{(s)}, \{x_1, \dots, x_T\}^{(s)}$ for $s = 1, \dots, S$ simulations to assess whether observing a single time series is enough to estimate the mean of the processes.

```r
## Example 2: weak dependence

## Simulate process y_t
# initialize TxS matrix to store result
# (each column is a simulated time series {y_1, ..., y_T}^(s) )
nsim <- 3
y_list <- matrix(rep(NA, nsim*t_max), nrow = t_max, ncol = nsim)
# simulation
for (sim in 1:nsim) {
  set.seed(sim+123)
  U0 <- rnorm(1, sd = 10)
  z  <- rnorm(t_max)
  y_list[,sim] <- U0 + 0.25*z
}
```

As the plot below shows, if we observe a single simulated time series we cannot correctly estimate the mean of the process $\mathbb{E}[y_t]$ using the sample mean $\bar{y}_T$. What the sample mean is actually estimating in this case is the *conditional* expctation of y_t given the specific draw of the random intercept U_0 in simulation s: $\mathbb{E}\left[y_t | U_0 = U_0^{(s)}\right] = U_0^{(s)}$. Notice that in this case, if we observe a new point for stochastic process y_t (green points in the plot) we are likely not able to forecast it correctly based on the previous observations of a single time series.

```r
# plot
plot.ts(y_list[,1], ylim = c(-25, 25), main = "Realizations of non-ergodic series", ylab = "y")
lines(y_list[,2], col = "steelblue")
lines(y_list[,3], col = "tomato")
for (point in 1:100){
  points(x = 501, rnorm(1, 0, 10)+rnorm(1, 0, 1), col = "darkgreen", pch = 4)
}
abline(h = 0, col = "black", lwd = 2, lty = 2)
```

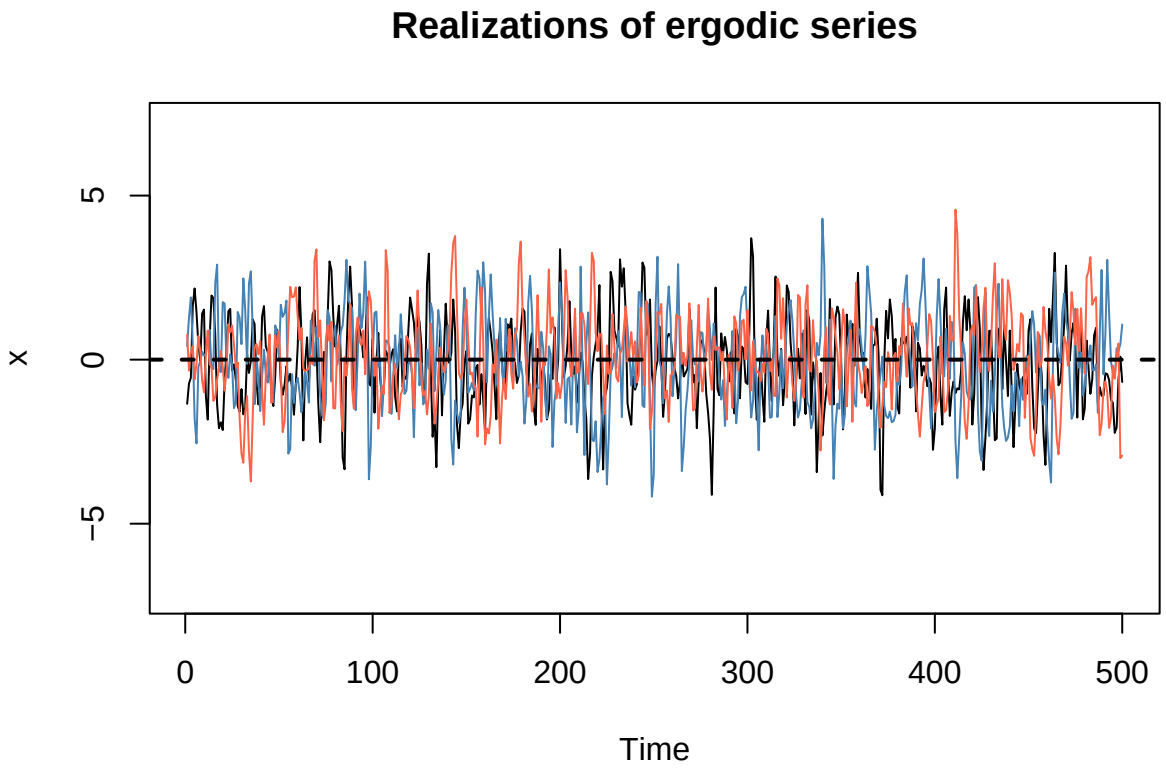


We now turn to process x_t.

```
## Simulate process x_t
# initialize object to store result
nsim <- 10000
x_list <- matrix(rep(NA, nsim*t_max), nrow = t_max, ncol = nsim)
# simulation
for (sim in 1:nsim) {
  set.seed(sim+123)
  z  <- rnorm(t_max+1)
  x_list[,sim] <- z[2:(t_max+1)] + z[1:t_max]
}
```

All the simulated time series have mean zero in this case. As a consequence, we can use the sample mean $\bar{x}_T$ obtained from a single time series to estimate the (unconditional) mean of the process $\mathbb{E}[x_t]$.

```
# plot
plot.ts(x_list[,1], ylim = c(min(x_list), max(x_list)),
        main = "Realizations of ergodic series", ylab = "x")
lines(x_list[,2], col = "steelblue")
lines(x_list[,3], col = "tomato")
abline(h = 0, col = "black", lwd = 2, lty = 2)
```



1.2 Asymptotic results

The problem with the series $\{y_t\}$ in the previous example is that the autocovariance function does not decay as the lag order increases. Loosely speaking, the series gets stuck in the trajectory given by the initial draw of U_0 and does not revert to the true mean of the stochastic process. The main consequence is that we cannot learn the mean of the process by taking the average of the observations in a single realized time series.

It is true in general that some conditions on the rate of decay of the autocovariance are required to recover the mean of the process. For example, the Law of Large Numbers (LLN) guarantees that the sample mean converges in probability to the true mean under the assumption that the autocovariances are absolutely summable:

$$\sum_{k=0}^{\infty} |\gamma_k| < \infty$$

Notice that in the case of $\{y_t\}$ above instead $\sum_{k=0}^{\infty} |\gamma_k| = \infty$. On the contrary, $\{x_t\}$ satisfies both the conditions of the LLN and of the Central Limit Theorem (CLT). The condition for the latter is that $\{\phi_k\}_{k=0}^{\infty}$ is absolutely summable in $x_t = \mu_x + \sum_{k=0}^{\infty} \phi_k z_{t-k} = z_t + z_{t-1}$, which is trivially verified. Therefore, since $\mu_x = 0$,

$$\sqrt{T}\, \bar{x}_T \xrightarrow{d} \mathcal{N}(0, \sigma^2_{LR}),$$

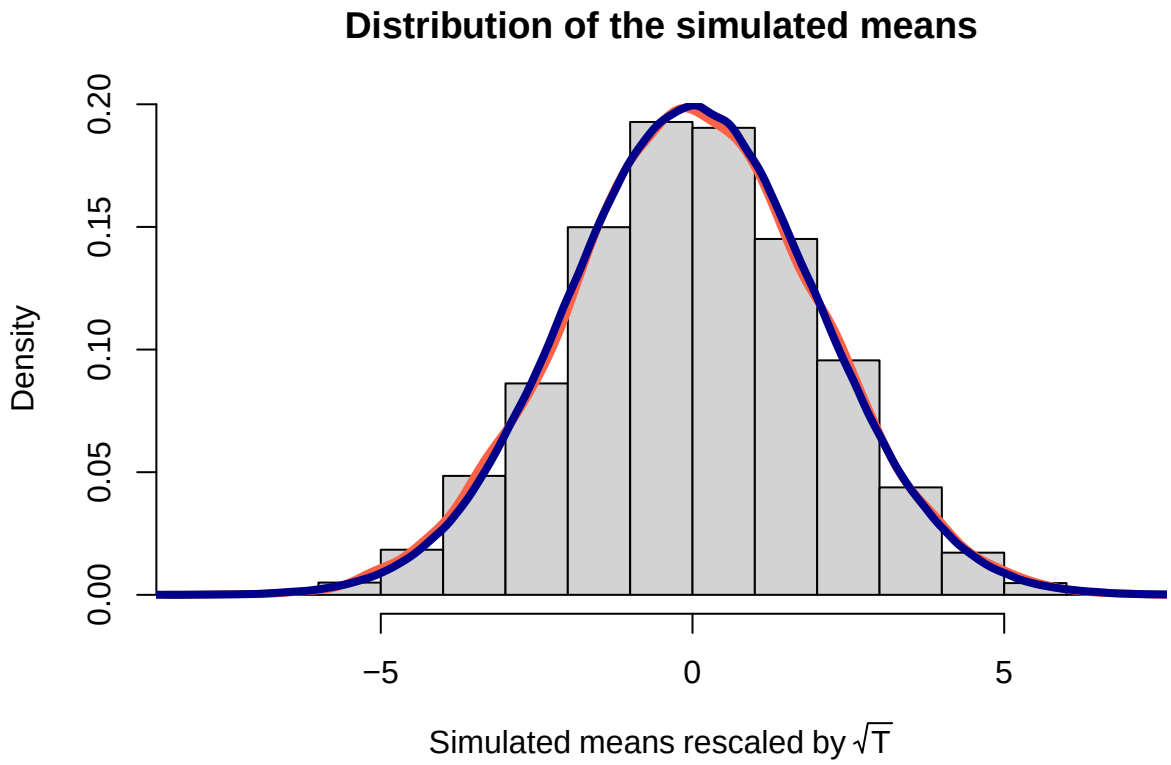
with $\sigma^2_{LR} = \sum_{k=-\infty}^{\infty} \gamma_k = \text{Var}(x_t) + 2\sum_{k=1}^{\infty} \gamma_k$.

```
## Example 3: central limit theorem
x_empirical <- x_list[,9]
x_means <- colMeans(x_list)
x_theory_variance <- 4  # long run variance
x_empir_variance <- var(x_empirical) + 2*(cov(x_empirical[1:(t_max-1)], x_empirical[2:t_max]))

rbind(empirical_variance = x_empir_variance,
```

```
      simulated_variance = var(x_means*sqrt(t_max)),
      theoretical_variance = x_theory_variance)
```

```
##                          [,1]
## empirical_variance   4.174614
## simulated_variance   4.050667
## theoretical_variance 4.000000
```

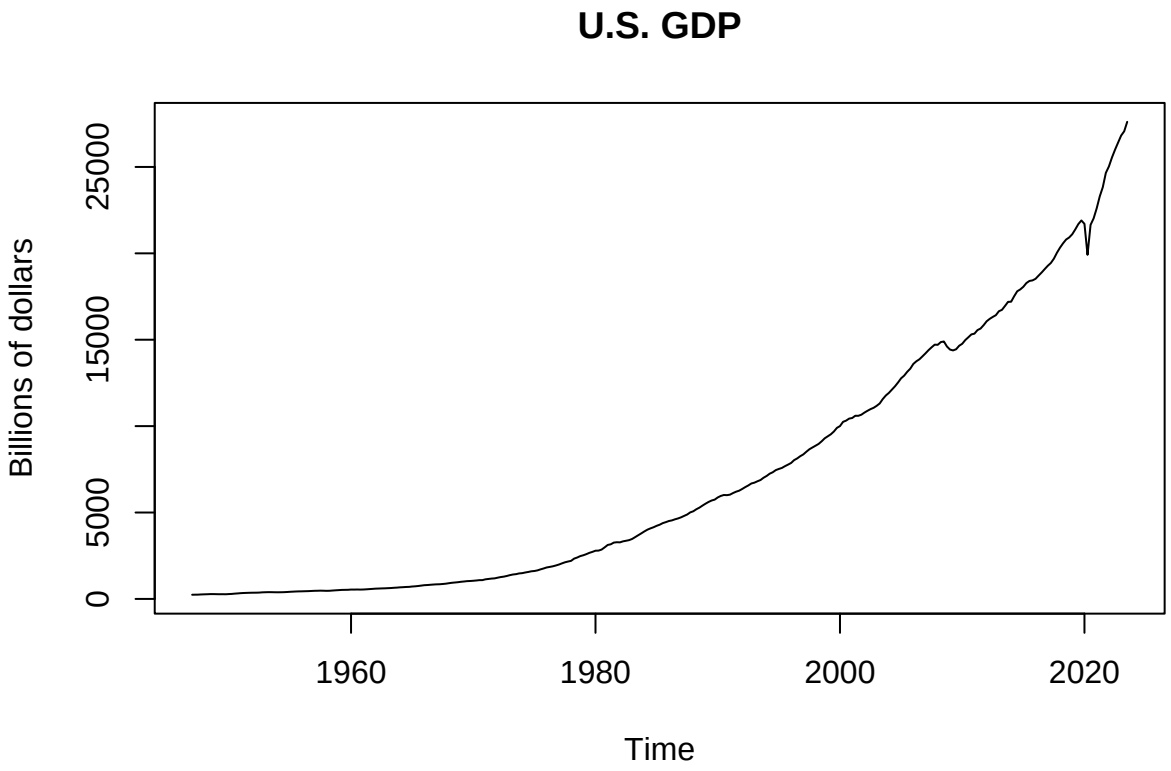
```
## Plot empirical distribution VS. theoretical distribution
hist(x_means*sqrt(t_max), breaks = 20, freq = FALSE,
     main = "Distribution of the simulated means",
     xlab = expression(Simulated ~ means ~ rescaled ~ by ~ sqrt(T)))
lines(density(x_means*sqrt(t_max)), lwd = 4, col = "tomato")
lines(density(rnorm(1e6, mean = 0, sd = sqrt(x_theory_variance))), lwd = 4, col = "darkblue")
```



1.3 Empirical moments and summary statistics

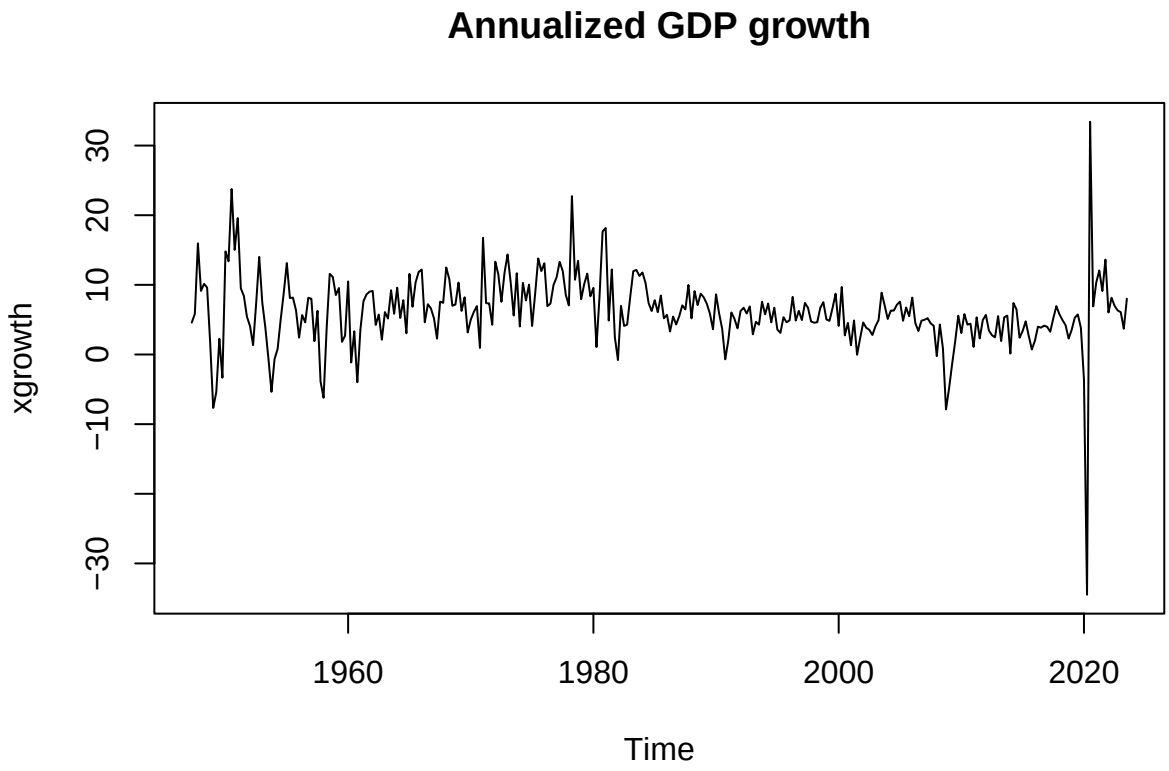
For this example we use the U.S. GDP data.

```
x <- read.csv("../data/us-gdp.csv")[,2]
x <- ts(x, start = c(1947, 1), frequency = 4)
t_max <- length(x)
plot.ts(x, main = "U.S. GDP", ylab = "Billions of dollars")
```



Consider the annualized quarterly growth rates:

```
# xgrowth <- ( (x[2:t_max]/x[1:(t_max-1)])^4 - 1 )*100
xgrowth <- 4*diff(log(x))*100
xgrowth <- ts(xgrowth, start = c(1947, 2), frequency = 4)
plot.ts(xgrowth, main = "Annualized GDP growth")
```



Below we compute some summary statistics for the time series of the U.S. GDP growth rates.

```
## Example 4: empirical moments and summary statistics of the GDP growth
library(moments)
rbind(
  mean     = mean(xgrowth),
  variance = var(xgrowth),
  skewness = skewness(xgrowth),
  kurtosis = kurtosis(xgrowth),
  min      = min(xgrowth),
  max      = max(xgrowth),
  above_5  = mean(xgrowth > 5),
  annualized_volatility = sqrt(4)*sd(xgrowth)
)
```

```
##                              [,1]
## mean                    6.1858847
## variance               26.5117326
## skewness               -0.9793909
## kurtosis               17.9597257
## min                   -34.4929551
## max                    33.4065902
## above_5                 0.6078431
## annualized_volatility  10.2979090
```

```
## Autocovariance function
gamma <- function(x, k) {
  k <- abs(k)
  t_max <- length(x)
  # (t_max-k)/t_max * cov(x[1:(length(x)-k)], x[(k+1):length(x)]) # for compatibility with acf()
  cov(x[1:(t_max-k)], x[(k+1):t_max])
```

```
}

## Autocorrelation
rho <- function(x, k) {gamma(x, k) / gamma(x, 0)}

## autocorrelation at different lags
sapply(0:12, rho, x = xgrowth)
```

```
##  [1]  1.000000000  0.262228567  0.254514380  0.097922439  0.030475252
##  [6] -0.016390183 -0.003186283  0.054980002  0.062358818  0.146571003
## [11]  0.169620140  0.143289793  0.096325249
```

Under the null hypothesis $H_0 : \rho = 0$, the sample autocorrelation is distributed as

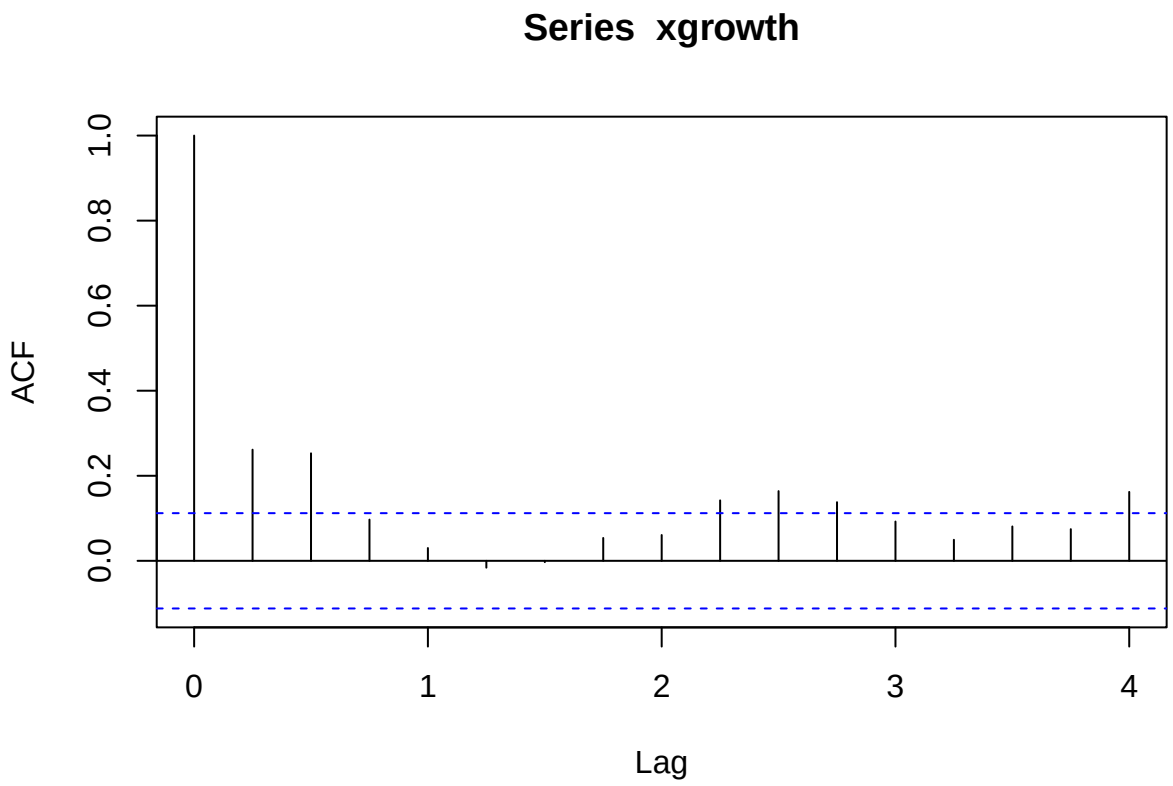
$$\sqrt{T}\ \hat{\rho} \xrightarrow{d} \mathcal{N}(0,1)$$

This means that the asymptotic variance of the estimator under the null is $1/T$. The plot reports the 95% confidence interval obtained as $\left(0 \pm \frac{z_{0.975}}{\sqrt{T}}\right)$.

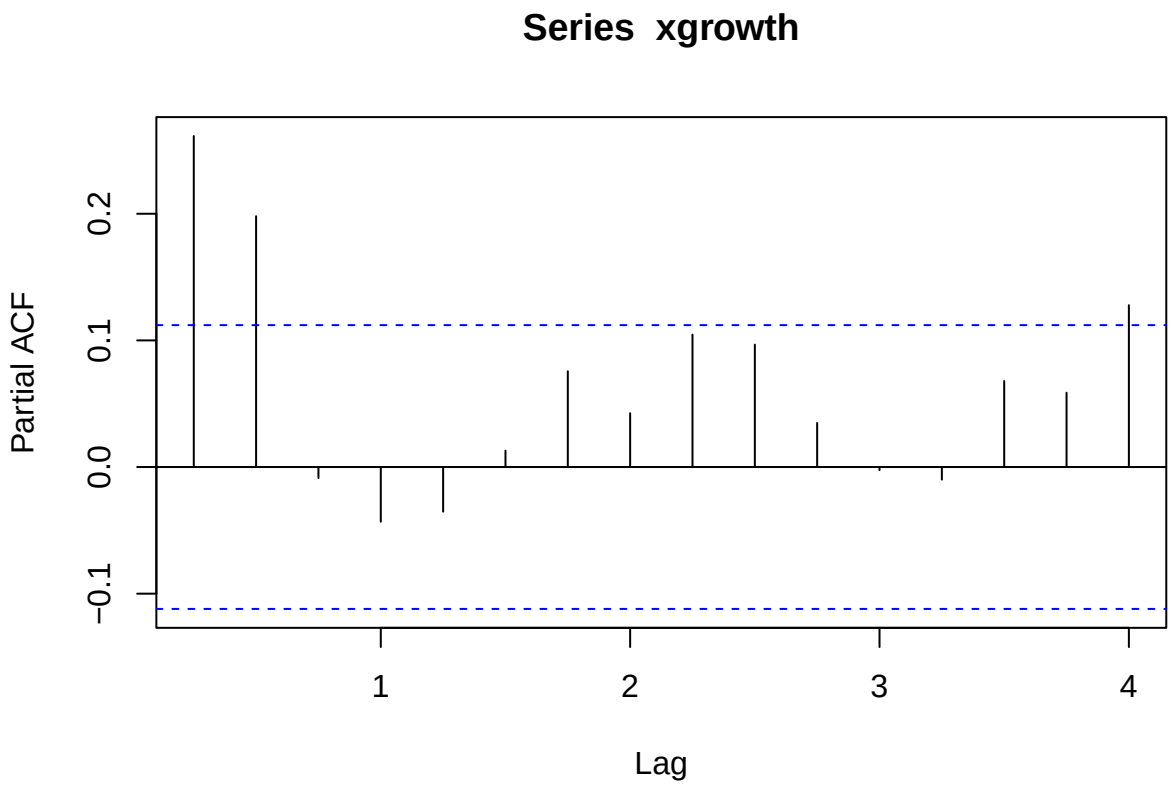
```
## confidence interval
qnorm(0.975)*(1/sqrt(t_max))
```

```
## [1] 0.1118611
```

```
## using built-in function for the autocorrelogram
acf(xgrowth, lag.max = 16)
```



```
## partial autocorrelation function
pacf(xgrowth, lag.max = 16)
```



1.4 Hypothesis testing

In this section we use three testing procedures on the GDP growth data: the Augmented Dickey-Fuller test for stationarity, the Jarque-Bera test for normality and the t-test for the mean of a process.

```
## Example 5: tests on GDP growth rates
library(tseries)

## stationarity: augmented Dickey-Fuller
adf.test(x)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  x
## Dickey-Fuller = 2.4611, Lag order = 6, p-value = 0.99
## alternative hypothesis: stationary
```

```
adf.test(xgrowth)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  xgrowth
## Dickey-Fuller = -5.9064, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

```r
## normality: Jarque-Bera
jarque.bera.test(xgrowth)
```

```
##
##  Jarque Bera Test
##
## data:  xgrowth
## X-squared = 2902.3, df = 2, p-value < 2.2e-16
```

```r
## mean zero: t-test
sigmaLR <- sum(sapply(-100:100, gamma, x = xgrowth))
t_stat  <- mean(xgrowth)/(sqrt(sigmaLR/t_max))
p_value <- (1-pnorm(abs(t_stat)))*2
# plot(density(rnorm(1e6)), xlim = c(-8, 8))
# abline(v = t_stat, col = "tomato", lwd = 2)

# compare with unadjusted variance
# t.test(xgrowth, mu = 0)
# mean(xgrowth)/(sqrt((t_max/(t_max-1))*var(xgrowth)/t_max)) # for compatibility with t-test()
t_stat_unadjasted <- mean(xgrowth)/(sqrt(var(xgrowth)/t_max))
p_value_unadjasted <- (1-pnorm(abs(t_stat_unadjasted)))*2

rbind(p_value_unadjasted = p_value_unadjasted,
      p_value = p_value)
```

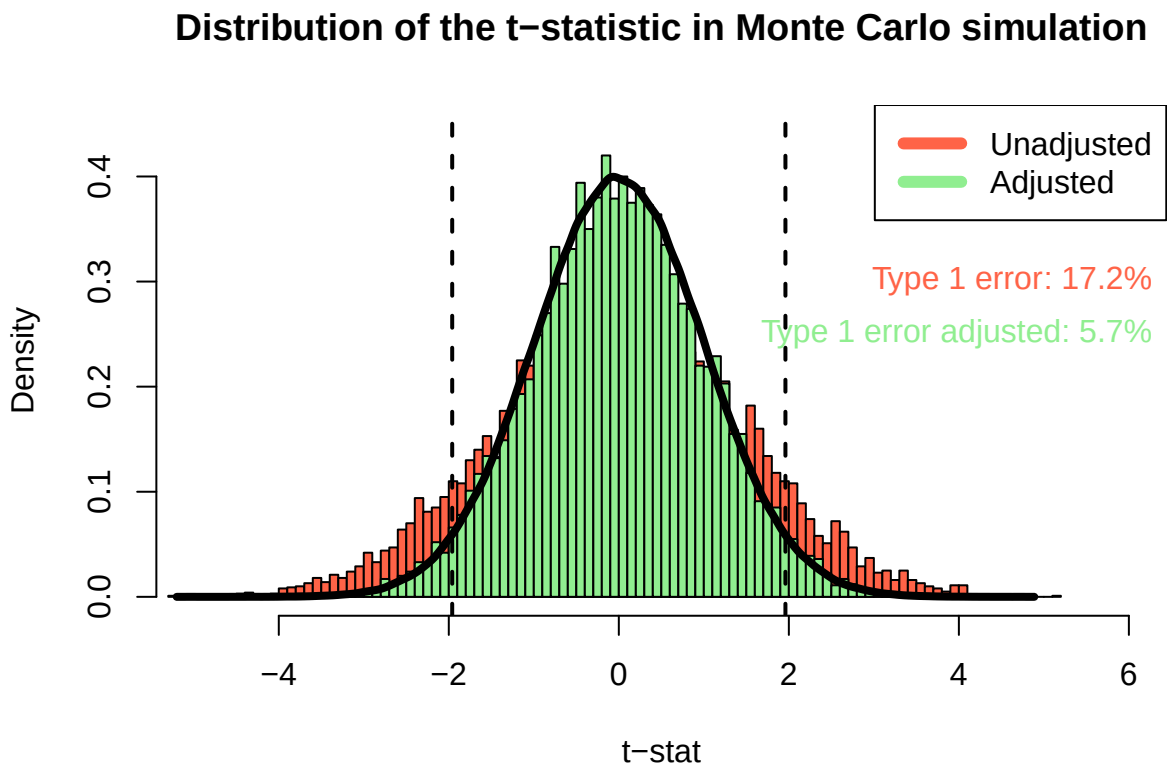
```
##                             [,1]
## p_value_unadjasted 0.000000e+00
## p_value            4.884981e-15
```

```r
# We can study when does the adjustment really matters
# using the results of the previous Monte Carlo simulation

## Get variance and long-run variance
sigma <- apply(x_list, 2, function(x) sqrt(gamma(x, k = 0)/length(x)))
sigmaLR <- apply(x_list, 2, function(x) sqrt(sum(sapply(-3:3, gamma, x = x))/length(x)))

## Get adjusted and unadjusted t-statistics
t_stat_unadjasted <- x_means/sigma
t_stat <- x_means/sigmaLR

## Compute percentage of type-1 errors in the simulation
type1error <- mean(abs(t_stat) > qnorm(0.975))
type1error_unadjusted <- mean(abs(t_stat_unadjasted) > qnorm(0.975))

rbind(
  type1error = type1error,
  type1error_unadjusted = type1error_unadjusted
)
```

```
##                           [,1]
## type1error              0.0569
## type1error_unadjusted 0.1716
```

```r
## plot distributions of adjusted and unadjusted t-statistics
hist(t_stat_unadjasted, breaks = 100, freq = FALSE, col = "tomato",
     ylim = c(0, 0.45), xlim = c(-5, 6), xlab = "t-stat",
     main = "Distribution of the t-statistic in Monte Carlo simulation")
```

```r
hist(t_stat, breaks = 100, freq = FALSE, add = TRUE, col = "lightgreen")
lines(density(rnorm(1e6, mean = 0, sd = 1)), lwd = 4, col = "black")
legend("topright", c("Unadjusted", "Adjusted"), col=c("tomato", "lightgreen"), lwd=6)
abline(v = qnorm(c(0.025, 0.975)), lty = 2, lwd = 2)
text(x=c(6.5, 6.5), y=c(0.3, 0.25),
     labels=c(paste0("Type 1 error: ", round(type1error_unadjusted*100, 1), "%"),
              paste0("Type 1 error adjusted: ", round(type1error*100, 1), "%")),
     col=c("tomato", "lightgreen"), pos = 2)
```



Chapter 2

Forecasting with time-series regression

This session covers a first introduction to forecasting the conditional mean of a time series. A forecasting procedure involves three main objects:

- A set of predictors and a target variable that we want to forecast
- A class of forecasting models that we consider
- A loss function to evaluate the accuracy of the forecasts

In this session we focus on the class of linear regression models estimated via least-squares. The loss function that we consider throughout this chapter is the quadratic loss function estimated using the Mean Squared Error (MSE) defined in the `R` function below.

```
## Function to compute the MSE. y = true value, f = forecast
mse <- function(y, f) {mean((y-f)^2)}
```

The first section presents a simulation exercise to show the finite-sample properties of the linear regression model under mis-specification. The second section is an empirical application to the problem of forecasting the U.S. unemployment rate using some relevant macroeconomic indicators.

2.1 Oracle inequality

Optional section, not covered in class

Given a forecasting problem, the optimal prediction rule is defined as the one that minimizes the theoretical risk:

$$f^*_{t+h|t} = \arg\min_{f \in \mathcal{F}} R(f_{t+h|t})$$

and the associated optimal risk is denoted by R^*. Under the square loss function, the optimal rule is the conditional expectation:

$$f^*_{t+h|t} = \mathbb{E}[y_{t+h}|x_t]$$

Notice that the exact functional form of the predictor is unknown: the conditional expected value could be of any linear or non-linear form. To make the theory of learning operational, we then have to restrict our attention to a class of prediction rules and pick the best rule from that class according to the limited amount of data available.

A commonly used class of predictors is that of the linear predictors. Consider a linear regression model for predicting the target variable y_{t+h}:

$$f_{t+h|t} = x_t'\theta.$$

where $f_{t+h|t}$ denotes the h-periods-ahead forecast. Irrespective of what is the true d.g.p., θ is the parameter that leads to the best forecasts *within the class of linear regression models*

$$\theta = \arg\min_{\theta \in \mathbb{R}^p} \mathbb{E}[L(y_{t+h}, f_{t+h|t}(\theta))] = R(\theta)$$

However, in practice we can only use a finite sample to learn the parameters of the linear model via *empirical* risk minimization:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} R_T(\theta)$$

In our case, $R_T(\theta) = \frac{1}{T-h-t_w+1} \sum_{t=t_w}^{T-h} (y_{t+h} - f_{t+h|t})^2 = MSE$, where t_w denotes the initial number of observations used to train the model.

So far, we have introduced three different measures of risk:

- R^*, the risk associated with the theoretical optimal predictor if the true d.g.p. were known
- $R(\theta)$, the risk associated with the optimal predictor within the parametric class considered for learning
- $R_T(\hat{\theta})$, the empirical risk obtained by minimization of the loss function using the available data sample

The regret (i.e. the difference between the optimal and the empirical risk) can be expressed in a way that links these three quantities:

$$R_T(\hat{\theta}) - R^* = \underbrace{[R(\theta) - R^*]}_{\text{Approximation error}} + \underbrace{\left[R_T(\hat{\theta}) - R(\theta)\right]}_{\text{Estimation error}}$$

We mainly focus on the estimation error. In particular, we are usually interested in establishing finite-sample, probabilistic bounds to this quantity. These bounds are helpful in assessing, for instance, how many observations a model needs to perform well and how its performance relates to the number of predictors. For linear prediction, the following oracle inequality holds: there exists a constant c such that

$$R(\hat{\theta}) \leq R(\theta) + c\frac{p}{T}, \quad \text{w.p. } 1 - 1/T$$

The example below shows that the inequality holds in a simulation setting. We generate 10^6 observations from the following process:

$$y_t = 0.6 \times y_{t-1} - \tanh(0.3 \times y_{t-1}) + 0.8 \times x_{1,t-1} + 0.2 \times x_{2,t-1} - 0.7 \times x_{3,t-1} + \varepsilon_t,$$

where $\varepsilon_t \sim \mathcal{N}(0, 0.5)$ and the variables $x_{1,t}, x_{2,t}, x_{3,t}$ are drawn independently from a standard normal distribution. We then consider the class of linear regression models,

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 x_{1,t-1} + \beta_3 x_{2,t-1} + \beta_3 x_{3,t-1} + u_t$$

Notice that the OLS estimator is the empirical risk minimizer in the case of a square loss. In the simulation setting, we can also obtain $R(\theta)$ by estimating the parameters of the regression using the entire simulated population. We can then draw many estimates of $R(\hat{\theta})$ by considering different subsamples of the simulated population and estimating the parameters of the model using OLS on the subsample. By doing so, we can simulate the distribution of the empirical risk and compare it against the overall minimum risk within the class of linear estimators to assess the oracle inequality. We repeat the exercise considering different sizes of the subsamples to show that the bound gets tighter as the sample size increases, as it must be for the inequality to have the oracle property.

```
## Example 1: simulation for oracle inequality
# build the simulated population dataset
N <- 3
T_pop <- 1e5

set.seed(455)
x_pop <- matrix(rnorm((T_pop+1)*N), ncol = N)
y_pop <- rep(0, T_pop)
for (t in 2:(T_pop+1)) {
  set.seed(456*t)
```

```r
  # non-linear d.g.p.
  y_pop[t] <- 0.6*y_pop[t-1] - tanh(0.3*y_pop[t-1]) +
              0.8*x_pop[t-1, 1] + 0.2*x_pop[t-1, 2] -
              0.7*x_pop[t-1, 3] + rnorm(1, sd = 0.5)
}
colnames(x_pop) <- paste0("x", 1:N)

# target is y(t+1). predictor is x(t)
d_pop <- as.data.frame(cbind(y = y_pop[2:(T_pop+1)] , x_pop[1:T_pop,] ))
```

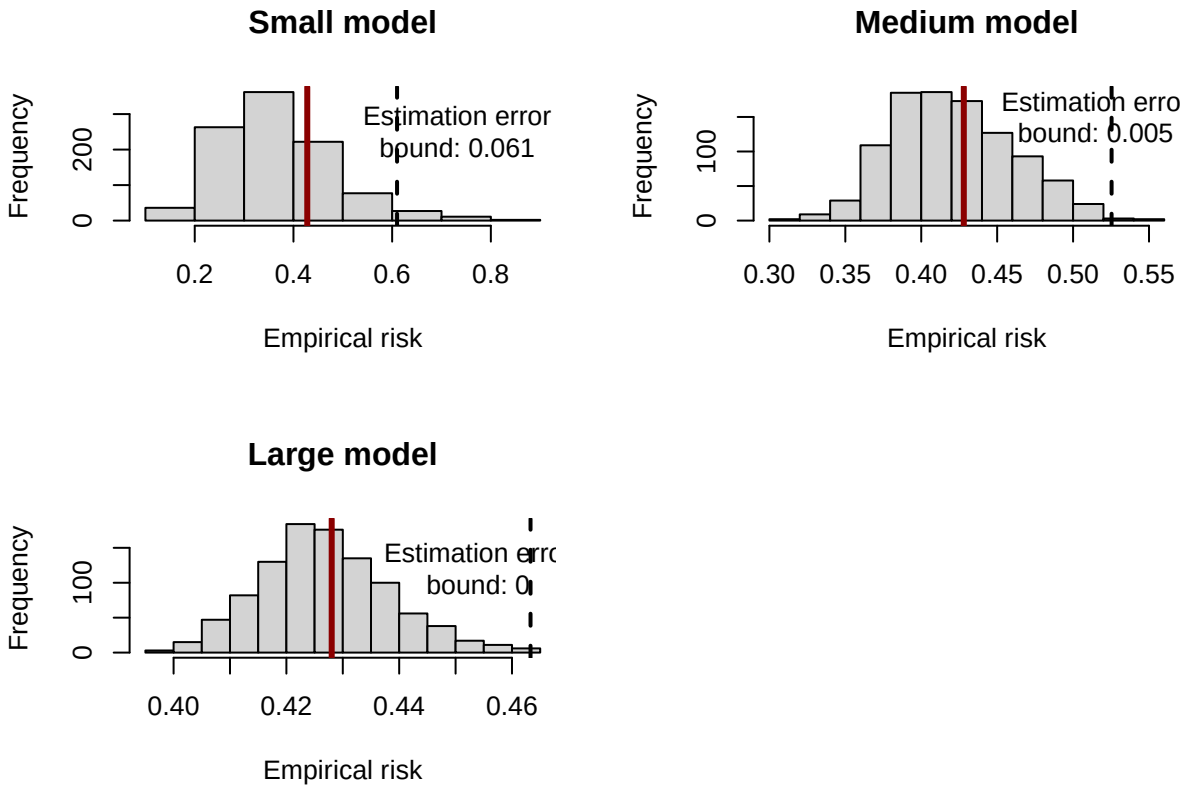
```r
# estimate the linear predictor y(t) = x(t-1)'b via OLS
model_pop <- lm(y ~ x1 + x2 + x3, data = d_pop)
summary(model_pop)
```

```r
## estimate the optimal risk and the population risk within the linear family
# optimal
f_optimal <- rep(NA, T_pop+1)
for (t in 1:T_pop) {
  f_optimal[t+1] <- 0.6*y_pop[t] - tanh(0.3*y_pop[t]) +
                    0.8*x_pop[t, 1] + 0.2*x_pop[t, 2] -
                    0.7*x_pop[t, 3]
}
f_optimal <- f_optimal[-1]
risk_star <- mse(d_pop[,1], f_optimal)

# linear family
risk_pop  <- mse(d_pop[,1], predict(model_pop))
```

```r
## Run the simulations to estimate a small-sample model many times
t_small  <- 30
t_medium <- 300
t_large  <- 3000
nsim <- 1000

empirical_risk <- list(
  model_small = rep(NA, nsim),
  model_medium = rep(NA, nsim),
  model_large = rep(NA, nsim)
)

for (s in 1:nsim) {
  set.seed(456*s)
  # pick index at random between 1e5 and 9e5
  idx <- sample(1:(T_pop-t_large), 1)
  # consider the t subsequent observations
  d_small  <- d_pop[idx:(idx+t_small), ]
  d_medium <- d_pop[idx:(idx+t_medium),]
  d_large  <- d_pop[idx:(idx+t_large), ]
  # estimate the model
  model_small <- lm(y ~ x1 + x2 + x3, data = d_small)
  model_medium <- lm(y ~ x1 + x2 + x3, data = d_medium)
  model_large <- lm(y ~ x1 + x2 + x3, data = d_large)
  # compute the MSE and store the results
  empirical_risk$model_small[s] <- mse(d_small[,1], predict(model_small))
  empirical_risk$model_medium[s] <- mse(d_medium[,1], predict(model_medium))
  empirical_risk$model_large[s] <- mse(d_large[,1], predict(model_large))
}
```

```r
# compute the constant C s.t. the bound risk_pop + C(P/t) is satisfied
# 1-1/t fraction of times
c_small <- quantile(empirical_risk$model_small, 1-1/t_small)
c_medium <- quantile(empirical_risk$model_medium, 1-1/t_medium)
c_large <- quantile(empirical_risk$model_large, 1-1/t_large)
```

```r
## report the distribution of the simulated empirical risks
par(mfrow = c(2,2))
# small
hist(empirical_risk$model_small, xlab = "Empirical risk", main = "Small model")
abline(v = c_small, lwd = 2, lty = 2)
abline(v = risk_pop, lwd = 3, lty = 1, col = "darkred")
text(x = c_small*1.2, y = 250, labels = paste0("Estimation error\nbound: ",
                                               round(c_small*N/t_small,3)))
# medium
hist(empirical_risk$model_medium, xlab = "Empirical risk", main = "Medium model")
abline(v = c_medium, lwd = 2, lty = 2)
abline(v = risk_pop, lwd = 3, lty = 1, col = "darkred")
text(x = c_medium*0.98, y = 150, labels = paste0("Estimation error\nbound: ",
                                                 round(c_medium*N/t_medium,3)))
# large
hist(empirical_risk$model_large, xlab = "Empirical risk", main = "Large model")
abline(v = c_large, lwd = 2, lty = 2)
abline(v = risk_pop, lwd = 3, lty = 1, col = "darkred")
text(x = c_large*0.98, y = 120, labels = paste0("Estimation error\nbound: ",
                                               round(c_large*N/t_large,3)))
```



2.2 Application: forecasting unemployment

```
library(sandwich) # for estimation of robust standard errors
library(lmtest)   # for coeftest
library(forecast) # for dm.test
```

We now consider an empirical application using the U.S. data from the FRED-MD dataset, a collection of 127 macroeconomic time series observed at a monthly frequency. To make sure that the series in the model are stationary, we transform the variables according to the recommendation of the authors, as summarised in the transformation code `tcode` in the first row of the dataset:

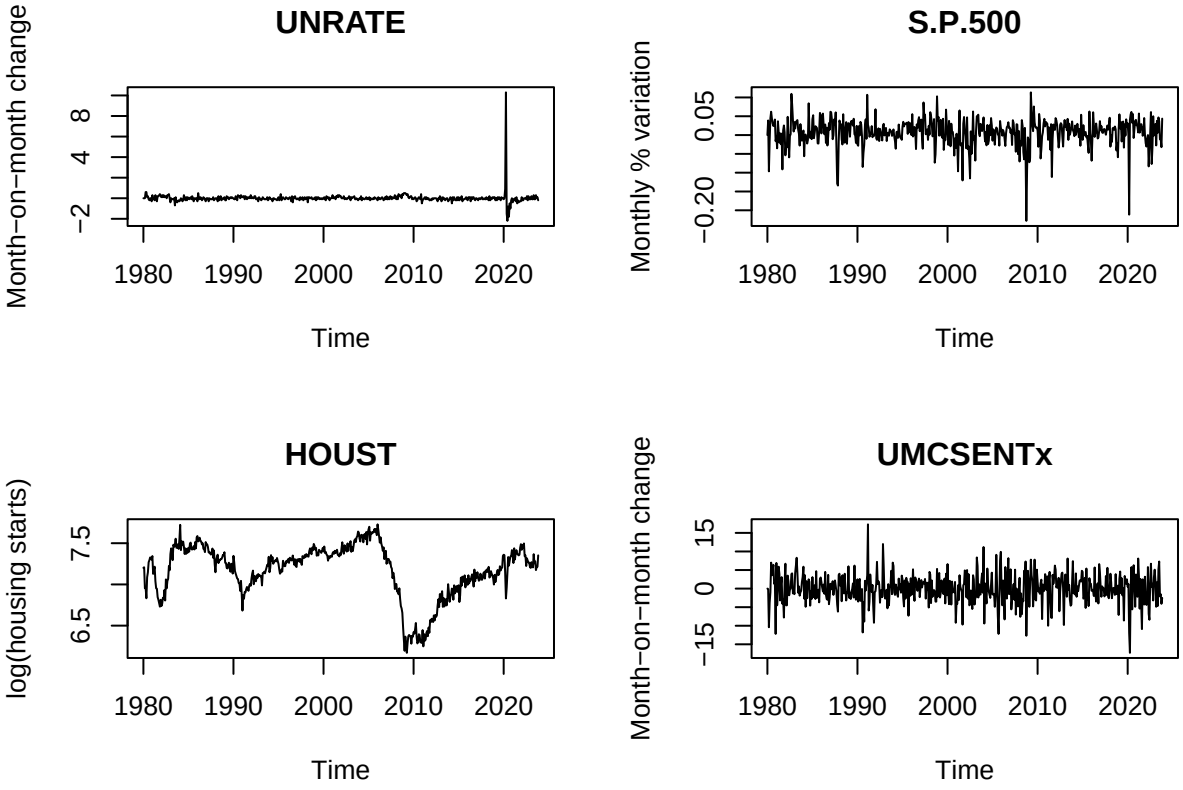
`tcode`	Transformation
1	No
2	Δx_t
3	$\Delta^2 x_t$
4	$\log(x_t)$
5	$\Delta \log(x_t)$
6	$\Delta^2 \log(x_t)$
7	$\Delta \left(\frac{x_t}{x_{t-1}} - 1 \right)$

```
## load FRED-MD data
fredmd <- read.csv("../data/current.csv")
dates  <- as.Date(fredmd$sasdate[-1],'%m/%d/%Y')
tcodes <- fredmd[1,-1]
d      <- fredmd[-1,-1]

# subset 1980-2024 period
data  <- d[dates >= '1980-01-01', ]
dates <- dates[dates >= '1980-01-01']
t_max     <- length(dates)

# choose variables
target <- "UNRATE"
predictors <- c('S.P.500','HOUST','UMCSENTx')
vars <- c(target, predictors)
data <- data[, vars]
tcodes <- tcodes[, vars]
```

```
## list of functions for transformation
transform_fredmd <- list(
  function(x) x,
  function(x) c(0, diff(x)),
  function(x) c(0, 0, diff(x, differences = 2)),
  function(x) log(x),
  function(x) c(0, diff(log(x))),
  function(x) c(0, 0, diff(log(x), differences = 2)),
  function(x) c(0, 0, diff( x[2:length(x)]/x[1:(length(x)-1)] - 1))
)
```

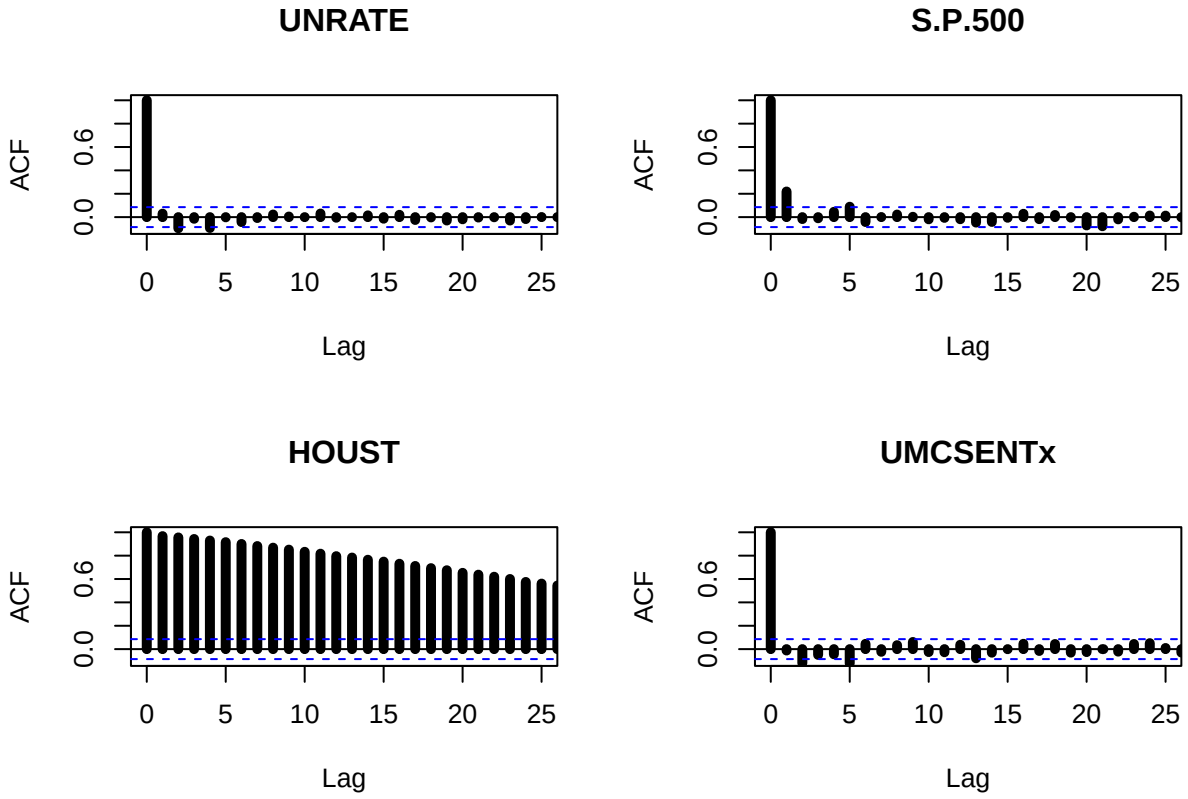
```
## save the transformed series separately
unrate <- transform_fredmd[[tcodes$UNRATE]](data$UNRATE)
sp500  <- transform_fredmd[[tcodes$S.P.500]](data$S.P.500)
house  <- transform_fredmd[[tcodes$HOUST]](data$HOUST)
sent   <- transform_fredmd[[tcodes$UMCSENTx]](data$UMCSENTx)
```

```
# plot
par(mfrow = c(2,2))
plot.ts(ts(unrate, start = c(1980, 1), freq = 12), main = "UNRATE", ylab = "Month-on-month change")
plot.ts(ts(sp500, start = c(1980, 1), freq = 12), main = "S.P.500", ylab = "Monthly % variation")
plot.ts(ts(house, start = c(1980, 1), freq = 12), main = "HOUST", ylab = "log(housing starts)")
plot.ts(ts(sent, start = c(1980, 1), freq = 12), main = "UMCSENTx", ylab = "Month-on-month change")
```



We can have a look to the autocorrelation properties of the series.

```
## ACF plot
par(mfrow = c(2,2))
acf(unrate, ylim=c(-0.1,1), lwd=5, xlim=c(0,25), main='UNRATE')
acf(sp500,  ylim=c(-0.1,1), lwd=5, xlim=c(0,25), main='S.P.500')
acf(house,  ylim=c(-0.1,1), lwd=5, xlim=c(0,25), main='HOUST')
acf(sent,   ylim=c(-0.1,1), lwd=5, xlim=c(0,25), main='UMCSENTx')
```



We now study some in-sample properties of the predictive regression:

$$\begin{aligned}\text{UNRATE}_t = \beta_0 + \beta_1\ \text{UNRATE}_{t-1} + \beta_2\ \text{HOUST}_{t-1}\ + \beta_3\ \text{SP500}_{t-1}\ + \beta_4\ \text{SENT}_{t-1} + \\ +\ \text{dummies}\ + \varepsilon_t.\end{aligned}$$

In particular, we compute the in-sample predictive fit of the model and we study the properties of the residuals.

```
## Fit predictive regression
# Create the dataframe for regression
data_regression <- data.frame(
  unrate     = unrate[2:t_max],
  unrate_lag = unrate[1:(t_max-1)],
  house      = house[1:(t_max-1)],
  sp500      = sp500[1:(t_max-1)],
  sent       = sent[1:(t_max-1)],
  d1 = (dates[2:t_max]=="2020-04-01")*1, # covid dummy variable
  d2 = (dates[2:t_max]=="2020-05-01")*1,
  d3 = (dates[2:t_max]=="2020-06-01")*1,
  d4 = (dates[2:t_max]=="2020-07-01")*1,
  d5 = (dates[2:t_max]=="2020-08-01")*1,
  d6 = (dates[2:t_max]=="2020-09-01")*1
)

# Fit OLS
pred_model <- lm(unrate ~ ., data = data_regression)
vcov_nw <- NeweyWest(pred_model, prewhite=F)

# BAD inference (unadjusted asymptotic variance)
coeftest(pred_model)
```

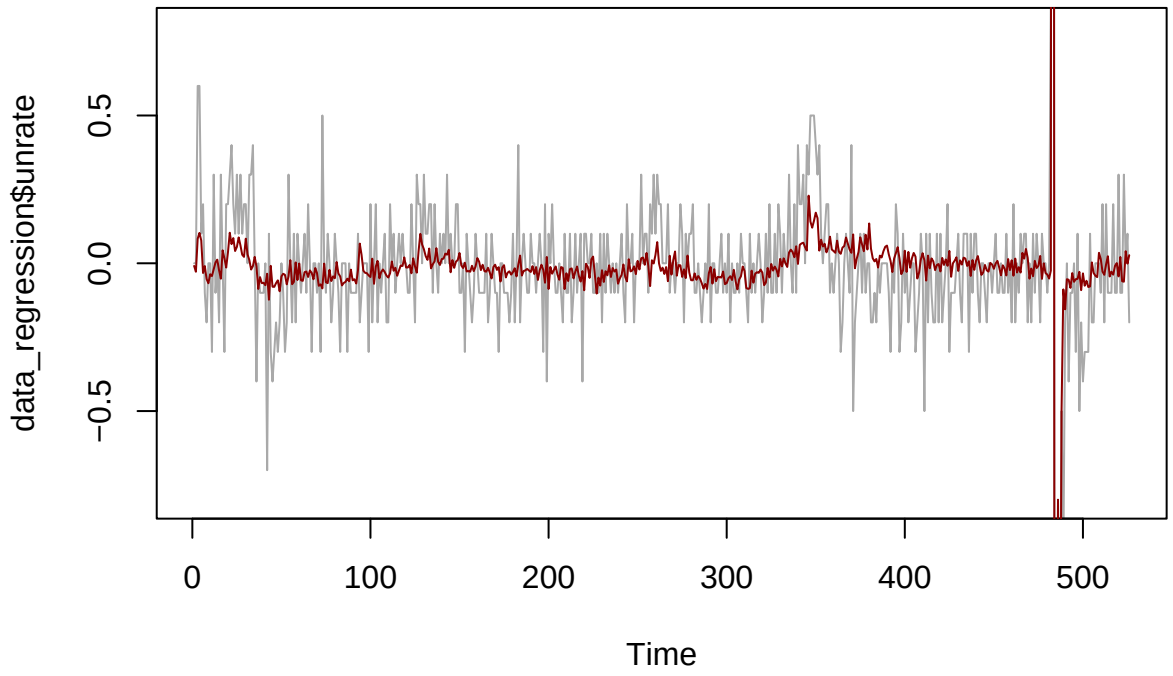
```
##
```

```
## t test of coefficients:
##
##               Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  0.6066299  0.1752768   3.4610 0.0005829 ***
## unrate_lag   0.1241838  0.0445711   2.7862 0.0055298 **
## house       -0.0854805  0.0244385  -3.4978 0.0005097 ***
## sp500       -0.4989494  0.2318030  -2.1525 0.0318242 *
## sent        -0.0021749  0.0020826  -1.0443 0.2968167
## d1          10.0608718  0.1859894  54.0938 < 2.2e-16 ***
## d2          -2.8193347  0.4924891  -5.7247 1.762e-08 ***
## d3          -1.9965983  0.1882808 -10.6044 < 2.2e-16 ***
## d4          -0.4795102  0.2010676  -2.3848 0.0174483 *
## d5          -1.6764097  0.1795464  -9.3369 < 2.2e-16 ***
## d6          -0.2339512  0.1927566  -1.2137 0.2254137
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

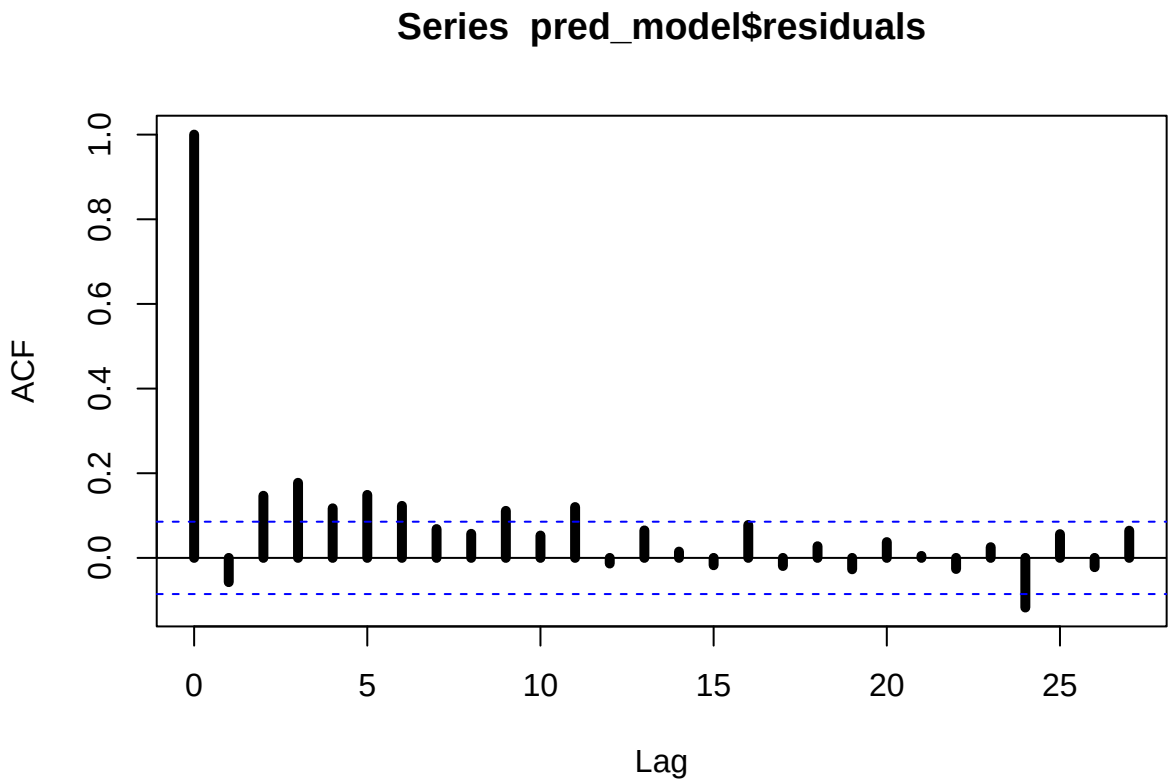
```
# Inference using the Newey West estimator
coeftest(pred_model, vcov_nw)
```

```
##
## t test of coefficients:
##
##               Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  0.6066299  0.3400340   1.7840   0.07501 .
## unrate_lag   0.1241838  0.0686069   1.8101   0.07087 .
## house       -0.0854805  0.0464922  -1.8386   0.06655 .
## sp500       -0.4989494  0.2553605  -1.9539   0.05125 .
## sent        -0.0021749  0.0024861  -0.8748   0.38207
## d1          10.0608718  0.0970859 103.6286 < 2.2e-16 ***
## d2          -2.8193347  0.7159484  -3.9379 9.351e-05 ***
## d3          -1.9965983  0.1022064 -19.5350 < 2.2e-16 ***
## d4          -0.4795102  0.1528230  -3.1377   0.00180 **
## d5          -1.6764097  0.0584554 -28.6785 < 2.2e-16 ***
## d6          -0.2339512  0.1251049  -1.8700   0.06205 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# True VS. predicted values
plot.ts(data_regression$unrate, ylim = c(-0.8, 0.8), col = "darkgrey")
lines(predict(pred_model), col = "darkred")
```




```r
# Analysis of residuals
acf(pred_model$residuals, lwd = 5)
```



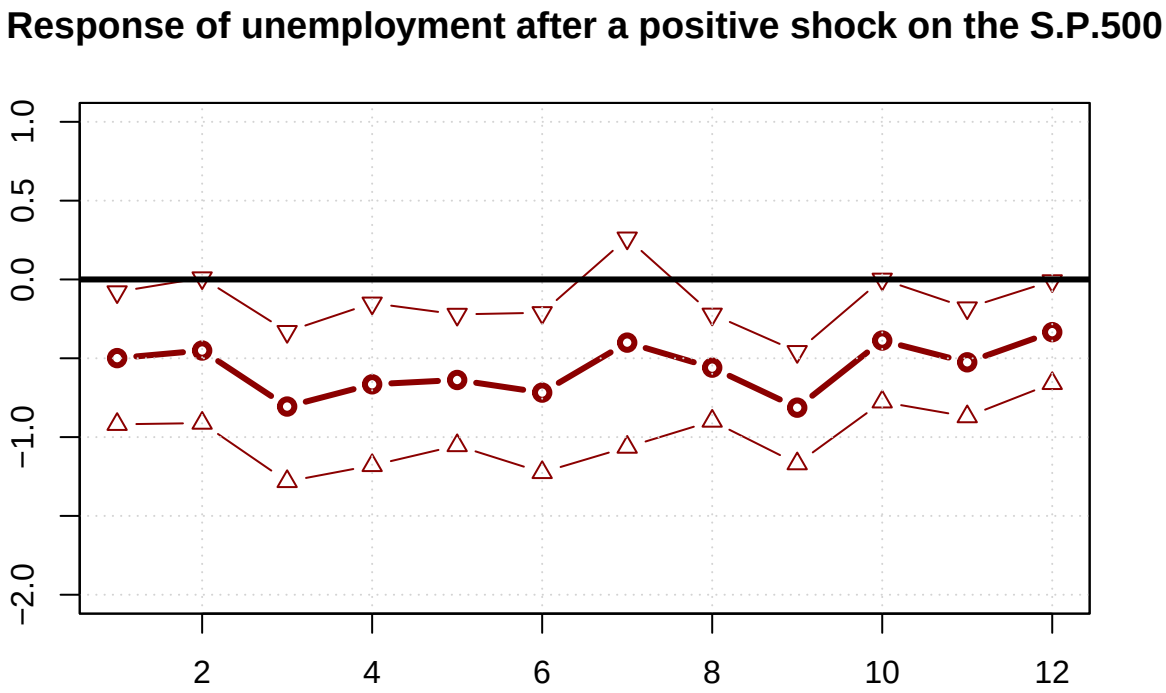
```
# Test for autocorrelation
Box.test(pred_model$residuals, lag=12)
```

```
##
##  Box-Pierce test
##
## data:  pred_model$residuals
## X-squared = 76.202, df = 12, p-value = 2.176e-11
```

We can assess the effect of an exogenous change in the predictors on the target variable over time by using the method of local projections. Here we show the effect of a positive shock in the S.P.500 index.

```
## Local projections for assessing the impact of a shock
# set number of horizons and initialize objects
H          <- 12
lp         <- rep(0,H)      # local projection
lp_confint <- matrix(0,H,2) # confidence intervals

# compute the impact after h = 1,...,H horizons
for(h in 1:H){
  # prepare dataset for forecasting h periods ahead
  data_h  <- cbind(
    data_regression[h:t_max, c("unrate", paste0("d", 1:6))],
    data_regression[1:(t_max-h+1), c("unrate_lag", "house", "sp500", "sent")]
    )

  # predict
  pred_model_h <- lm(unrate ~ ., data = data_h)

  # compute Newey-West estimate
  vcov_nw_h <- NeweyWest(pred_model_h, prewhite = F)

  # compute standard errors for the estimated coefficients
  confint_h <- coeftest(pred_model_h, vcov_nw_h)

  # store the estimates and the 90% confidence intervals
  lp[h]           <- confint_h['sp500','Estimate']
  lp_confint[h, ] <- lp[h] + confint_h['sp500','Std. Error']*c(-1,1)*qnorm(0.05)
}
```

```
## Plot the impulse response function
plot(1:H, lp,         lwd = 3, col = 'darkred', t = "b", ylim = c(-2,1),
     main = "Response of unemployment after a positive shock on the S.P.500",
     xlab = "", ylab = "")
lines(lp_confint[,1], lwd = 1, col = 'darkred', t = "b", pch  = 25)
lines(lp_confint[,2], lwd = 1, col = 'darkred', t = "b", pch  = 24)
grid()
box()
abline(h=0,lwd=3)
```



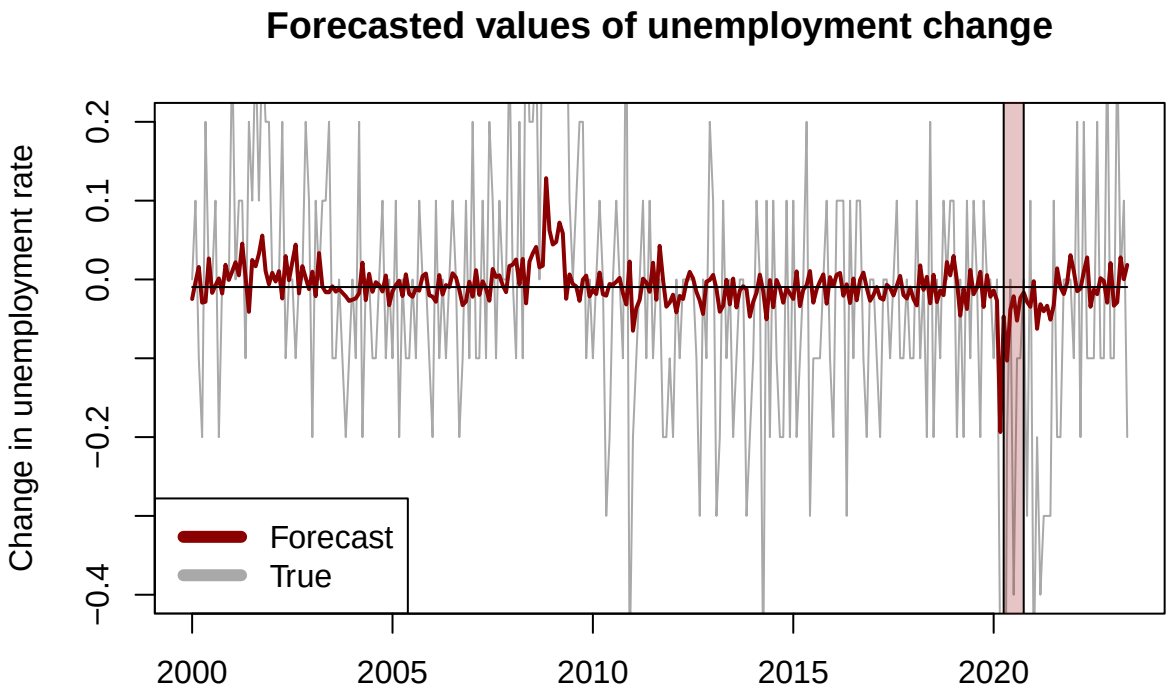
We now proceed to evaluate the forecasts of the model. To do so, we split the data into a training set (observations up to 1999:12) and a testing set (starting in 2000:01). We estimate the parameters of the model on the training data and then make predictions based on the value of the predictors and the estimated coefficients. We compare the forecasts against the sample mean using the MSE. The Diebold-Mariano test suggests that the regression method has a superior forecasting performance than the sample mean. However, this result depends on the specification of the regression model and the predictive ability of the model is still pretty poor.

```
## Forecast evaluation
# split training set (1980-2000) and testing set (2000-2024)
in_sample  <- (dates[2:t_max] < as.Date('2000-01-01'))
out_sample <- (dates[2:t_max] >= as.Date('2000-01-01') &
                 ( dates[2:t_max] < as.Date('2020-03-01') |
                     dates[2:t_max] >= as.Date('2020-09-01') ))
data_train <- data_regression[in_sample,]
data_test <- data_regression[out_sample,]
dates_test <- (dates[2:t_max])[out_sample]

# fit predictive models
model1 <- lm(unrate ~ unrate_lag + sp500 + sent + house, data = data_train)
model2 <- lm(unrate ~ unrate_lag + sp500 + sent,         data = data_train)
model3 <- lm(unrate ~ unrate_lag + sp500,                data = data_train)

# get predictions
y_hat1 <- predict(model1, newdata = data_test)
y_hat2 <- predict(model2, newdata = data_test)
y_hat3 <- predict(model3, newdata = data_test)

# compute benchmark prediction (sample mean)
y_benchmark  <- rep(mean(data_train$unrate), sum(out_sample))

# compute MSE
mse_model1     <- mse(data_test$unrate, y_hat1)
```

```r
mse_model2     <- mse(data_test$unrate, y_hat2)
mse_model3     <- mse(data_test$unrate, y_hat3)
mse_benchmark  <- mse(data_test$unrate, y_benchmark)

# results
round(rbind(
  mse_4vars = mse_model1,
  mse_3vars = mse_model2,
  mse_2vars = mse_model3,
  mse_mean  = mse_benchmark,
  R2_2vars  = 1 - mse_model3/mse_benchmark
), 4)
```

```
##             [,1]
## mse_4vars 0.0536
## mse_3vars 0.0302
## mse_2vars 0.0296
## mse_mean  0.0318
## R2_2vars  0.0677
```

```r
# convert to ts object to have dates in plot
y_true <- ts(data_test$unrate, start = c(2000, 1), frequency = 12)
y_hat3 <- ts(y_hat3, start = c(2000, 1), frequency = 12)
y_benchmark <- ts(y_benchmark, start = c(2000, 1), frequency = 12)

# plot
plot.ts(y_true, col = "darkgrey", ylim = c(-0.4, 0.2),
        main = "Forecasted values of unemployment change",
        ylab = "Change in unemployment rate",
        xlab = "")
lines(y_hat3, col = "darkred", lwd = 2)
lines(y_benchmark)
rect(2020.25,-1,2020.75,1,col = rgb(0.6,0.1,0.1,1/4))
legend("bottomleft", c("Forecast", "True"), col=c("darkred", "darkgrey"), lwd=6)
```

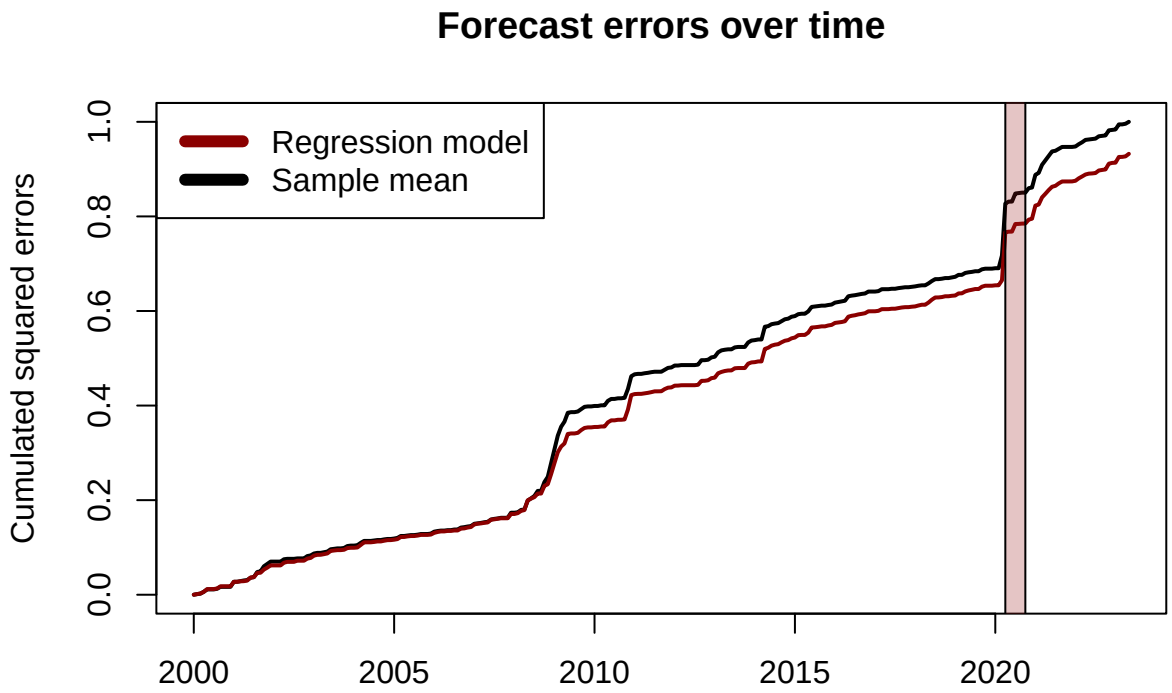


It is common to report the plot of the normalized cumulated squared errors to assess if one model has a superior forecasting ability over all the testing period:

$$\text{Cumulated error}_t = \frac{\sum_{\tau=t_w+1}^{t} e_\tau^2}{\sum_{\tau=t_w+1}^{T} \bar{e}_\tau^2}$$

where $e_\tau, \bar{e}_\tau$ denote respectively the forecast error of the model and the benchmark at time τ, and t_w is the number of periods in the training set.

```
## Plot the forecast errors over time
# compute the forecast errors
err_model <- y_true-y_hat3
err_benchmark <- y_true-y_benchmark
# compute the (normalized) cumulated squared errors and convert to ts object
err2_model     <- ts(cumsum((err_model)^2), start = c(2000, 1), frequency = 12)
err2_benchmark <- ts(cumsum((err_benchmark)^2), start = c(2000, 1), frequency = 12)
err2_constant  <- sum((y_true-y_benchmark)^2)
# plot the cumulated squared errors over time
plot.ts(err2_benchmark/err2_constant, lwd = 2,
        ylab = "Cumulated squared errors",
        main = "Forecast errors over time",
        xlab = "")
lines(err2_model/err2_constant, col = "darkred", lwd = 2)
rect(2020.25,-1,2020.75,1.5,col = rgb(0.6,0.1,0.1,1/4))
legend("topleft", c("Regression model", "Sample mean"), col=c("darkred", "black"), lwd=6)
```




```
## Diebold-Mariano test for predictive performance
# one-sided test:
# - H0: err_benchmark = err_model
# - H1: err_benchmark > err_model
dm.test(err_benchmark, err_model, alternative = "greater")
```

```
##
##  Diebold-Mariano Test
##
## data:  err_benchmarkerr_model
## DM = 2.4995, Forecast horizon = 1, Loss function power = 2, p-value =
## 0.006504
## alternative hypothesis: greater
```

Chapter 3

Large-dimensional methods for forecasting

In this session we compare the performances of four forecasting methods to predict the monthly change in the U.S. unemployment rate using more than 100 variables from the FRED-MD dataset. In order to deal with the large amount of data, we consider methods based on principal component analysis, penalized likelihood and regression trees.

3.1 Preprocess FRED-MD

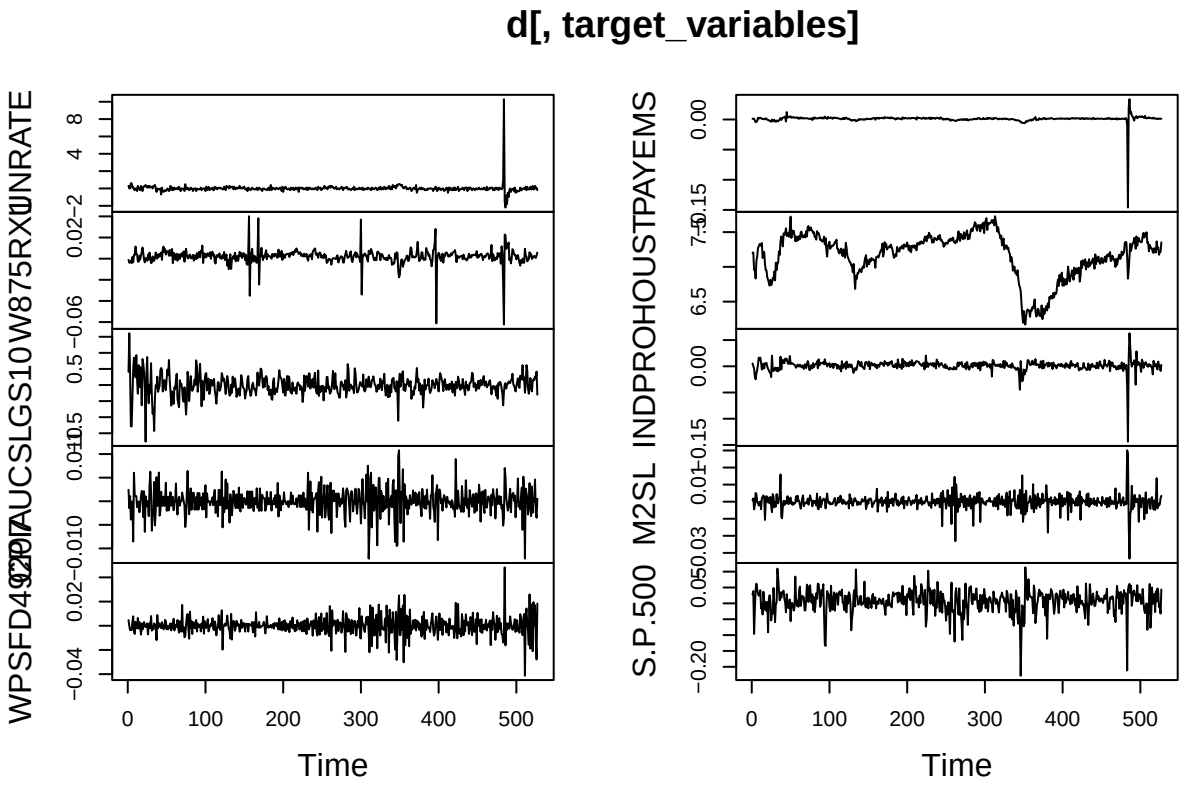
The preprocessing of the data is very similar to that of the previous session. The only difference is that we now transform all the series in the dataset using the list of transformation functions of the previous session.

For simplicity, here we remove the columns with missing values. An alternative could be to impute the missing values using some regression technique. However, it is important to notice that in the latter case we should impute the missing values at each step of the forecast evaluation to make sure that no future information is included in the forecasts.

```
## load data
fredmd <- read.csv("../data/current.csv")
dates  <- as.Date(fredmd$sasdate[-1],'%m/%d/%Y')
target_variables <- c("UNRATE", "W875RX1", "GS10", "CPIAUCSL", "WPSFD49207", "PAYEMS", "HOUST", "INDPRO", "M2SL"

## get tcodes
tcodes <- fredmd[1,-1]
d      <- fredmd[-1, -1]
```

```
## Transform variables
# transformation functions
transform_fredmd <- list(
  function(x) x,
  function(x) c(0, diff(x)),
  function(x) c(0, 0, diff(x, differences = 2)),
  function(x) log(x),
  function(x) c(0, diff(log(x))),
  function(x) c(0, 0, diff(log(x), differences = 2)),
  function(x) c(0, 0, diff( x[2:length(x)]/x[1:(length(x)-1)] - 1))
)

# transform series one by one
for (j in 1:ncol(d)) {
  tcode <- tcodes[[j]]
  d[,j] <- transform_fredmd[[tcode]](d[,j])
}

## Subset and remove columns with NAs
d <- d[dates >= '1980-01-01', ]
```

```
dates <- dates[dates >= '1980-01-01']
idx_na <- (colSums(is.na(d))==0)
d <- d[, idx_na]
plot.ts(d[,target_variables])
```



3.2 Forecasting algorithms

Below we define the forecasting algorithms that we use in the forecast evaluation.

Notice that at each step of the forecast evaluation we rescale the data so that each variable has mean zero and unit variance. This step is very important for many machine learning algorithms. If the variables have a different magnitude, the optimization procedure involved tend to over-focus on the features with higher magnitude, while we want to exploit the variability of each feature irrespectively of its scale.

Also, we do not explicitly account for outliers in the forecasting algorithms, and we simply discard the Covid period during the evaluation of the results. In this sense, the procedure will tend to benefit those methods that can automatically deal better with the presence of outliers in the data used for estimation.

```
## Principal Component Regression (PCR)
pcr <- function(d, target, n_components, horizon) {

  ## Scale the data
  d_scaled <- scale(d)
  d_mean <- attr(d_scaled, "scaled:center")
  d_sd <- attr(d_scaled, "scaled:scale")

  ## get the target
  target_idx <- which(colnames(d) == target)
  y <- d_scaled[,target_idx]
  ## get the principal components of the other variables
```

```
  x <- d_scaled[,-target_idx]
  eigen_list <- prcomp(x, center=FALSE)
  f <- eigen_list$x[,1:n_components]

  ## define the dataframes
  t_max <- length(y)
  d_pcr <- as.data.frame(cbind(
    y     = y[(horizon+1):t_max],
    y_lag =  y[1:(t_max-horizon)],
    f[1:(t_max-horizon),])
  )

  ## fit
  model <- lm(y ~ ., data = d_pcr)

  ## forecast
  pred <- predict(model, newdata = d_pcr[nrow(d_pcr), ])

  ## output
  pred*d_sd[target] + d_mean[target]
}

# pcr(d, "INDPRO", 4, 1)
```

```
## Penalized regression: LASSO (alpha = 1) and ridge (alpha = 0)
library(glmnet)
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
penalized_reg <- function(d, target, horizon, alpha = 1, nlambda = 100) {

  ## Scale the data
  d_scaled <- scale(d)
  d_mean <- attr(d_scaled, "scaled:center")
  d_sd <- attr(d_scaled, "scaled:scale")

  ## get the target and predictors
  t_max <- nrow(d)
  y <- d_scaled[(horizon+1):t_max, target]
  x <- d_scaled[1:(t_max-horizon), ] # include also lagged target variable

  ## fit the model
  model <- glmnet(x = x, y = y, family = "gaussian",
                  alpha = alpha, nlambda = nlambda)

  ## forecast
  pred <- predict(model, newx = x[nrow(x),])

  ## Output
  pred*d_sd[target] + d_mean[target]
}

# penalized_reg(d, "INDPRO", 1, 1) # one value for each lambda
```

```
## Random forest
library(randomForest)
forest <- function(d, target, horizon, seed = 1, ntree = 100) {

  ## Scale the data
  d_scaled <- scale(d)
  d_mean <- attr(d_scaled, "scaled:center")
  d_sd <- attr(d_scaled, "scaled:scale")

  ## get the target and predictors
  t_max <- nrow(d)
  y <- d_scaled[(horizon+1):t_max, target]
  x <- d_scaled[1:(t_max-horizon), ] # include also lagged target variable

  ## fit the model
  set.seed(seed)
  model <- randomForest(x = x, y = y, ntree = ntree)

  ## forecast
  pred <- predict(model, newdata = x[nrow(x),])

  ## Output
  pred*d_sd[target] + d_mean[target]
}

# forest(d, "INDPRO", 1)
```

3.3 Forecast evaluation

We can now run the algorithms and compare their forecasting performances over the 2002-2020 period.

```
## Initialize objects
forecast_list <- list(
  pcr = c(),
  ridge = list(),
  lasso = list(),
  forest = c()
)
# choose target variable and forecast horizon
target <- "UNRATE"
horizon <- 1

# pick the initial window for training and choose the test period
initial_window <- sum(dates < "2002-01-01")
idx_eval <- which(((dates < "2020-02-01")) &
                    dates >= "2002-01-01") - initial_window

# total steps of the forecast evaluation and selection of the test data
n_steps <- nrow(d) - initial_window - horizon + 1
data_test <- d[(initial_window+horizon):nrow(d), target]
```

```
## expanding window forecast evaluation
for (s in 1:n_steps) {
  cat("\rStep", s, "of", n_steps)
  data_train <- d[1:(initial_window+s-1),]

  forecast_list$pcr[s] <- pcr(data_train, target,
```

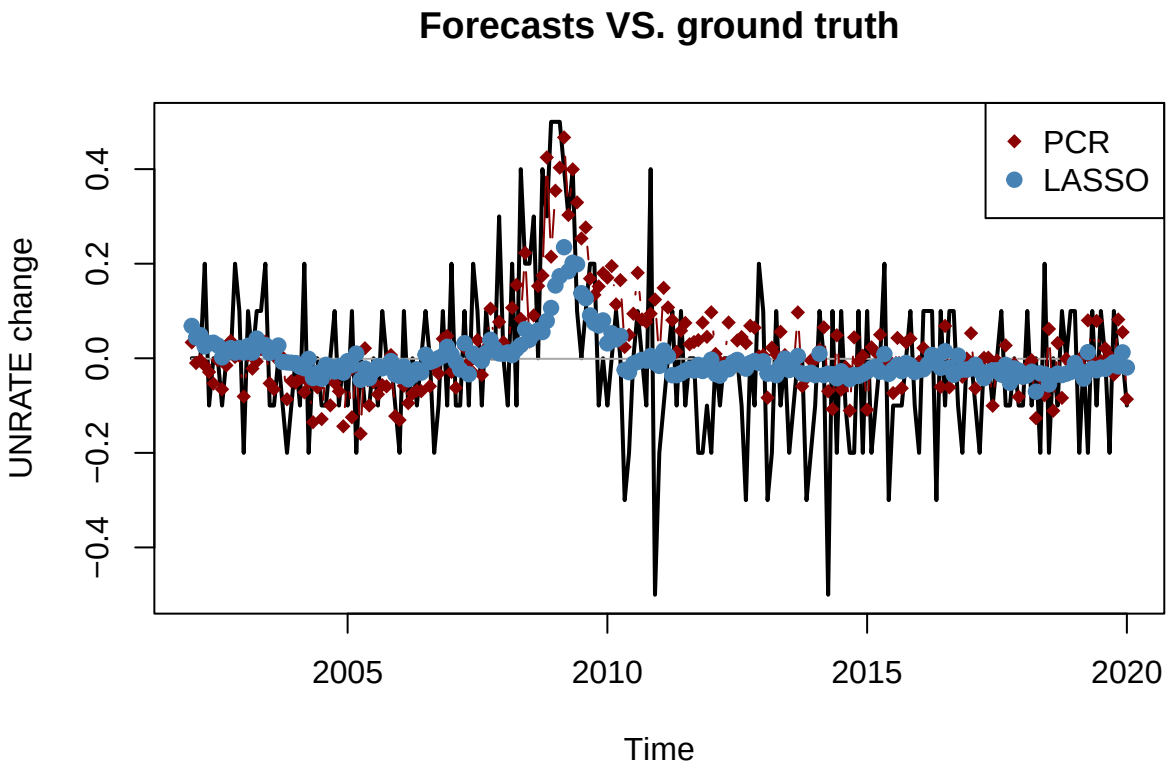
```
                                n_components = 4, horizon = horizon)
  forecast_list$ridge[[s]] <- penalized_reg(data_train, target,
                                            horizon = horizon, alpha = 0)
  forecast_list$lasso[[s]] <- penalized_reg(data_train, target,
                                            horizon = horizon, alpha = 1)
  forecast_list$forest[s] <- forest(data_train, target,
                                    horizon = horizon,
                                    ntree = 100)
}
```

Remark: here we just pick one value of λ among the many that we used for estimating ridge and LASSO regressions. A more rigourous approach would consist in using either information criteria or cross validation to pick the optimal λ. We omit this part for simplicity of exposition.

```
## Comppute the RMSE of the predictions
rmse <- function(y, f) {sqrt(mean((y-f)^2))}

pred_ridge <- sapply(forecast_list$ridge, function(x) x[30])
pred_lasso <- sapply(forecast_list$lasso, function(x) x[8])
pred_mean  <- rep(mean(d[1:initial_window, target]), n_steps)

rbind(
  mean   = rmse(data_test[idx_eval], pred_mean[idx_eval]),
  pcr    = rmse(data_test[idx_eval], forecast_list$pcr[idx_eval]),
  forest = rmse(data_test[idx_eval], forecast_list$forest[idx_eval]),
  ridge  = rmse(data_test[idx_eval], pred_ridge[idx_eval]),
  lasso  = rmse(data_test[idx_eval], pred_lasso[idx_eval])
)
```

```
##             [,1]
## mean   0.1600012
## pcr    0.1513172
## forest 0.1676554
## ridge  0.1475805
## lasso  0.1416051
```

```
## Plot the forecasted values
ground_truth   <- ts(data_test[idx_eval], start = c(2002, 1), freq = 12)
pred_benchmark <- ts(pred_mean[idx_eval], start = c(2002, 1), freq = 12)
pred_pcr       <- ts(forecast_list$pcr[idx_eval], start = c(2002, 1), freq = 12)
pred_lasso     <- ts(pred_lasso[idx_eval], start = c(2002, 1), freq = 12)
plot.ts(ground_truth, type = "l", lwd = 2,
        main = "Forecasts VS. ground truth",
        ylab = "UNRATE change")
lines(pred_benchmark, type = "l", col = "darkgrey")
lines(pred_pcr, type = "b", col = "darkred", pch = 18)
lines(pred_lasso, type = "b", col = "steelblue", pch = 19)
legend("topright", c("PCR", "LASSO"), col = c("darkred", "steelblue"),
       pch = c(18, 19))
```




```
## Diebold-Mariano test
library(forecast)
err_benchmark <- data_test[idx_eval] - pred_mean[idx_eval]
err_lasso <- data_test[idx_eval] - pred_lasso[idx_eval]
err_pcr <- data_test[idx_eval] - forecast_list$pcr[idx_eval]

# LASSO
dm.test(err_benchmark, err_lasso, alternative = "greater")
```

```
##
##  Diebold-Mariano Test
##
## data:  err_benchmarkerr_lasso
## DM = 3.9017, Forecast horizon = 1, Loss function power = 2, p-value =
## 6.381e-05
## alternative hypothesis: greater
```

```
# PCR
dm.test(err_benchmark, err_pcr, alternative = "greater")
```

```
##
##  Diebold-Mariano Test
##
## data:  err_benchmarkerr_pcr
## DM = 1.0484, Forecast horizon = 1, Loss function power = 2, p-value =
## 0.1478
## alternative hypothesis: greater
```

```r
## plot the cumulated squared errors over time
# compute the (normalized) cumulated squared errors and convert to ts object
err2_benchmark <- ts(cumsum((err_benchmark)^2), start = c(2002, 1), frequency = 12)
err2_pcr       <- ts(cumsum((err_pcr)^2), start = c(2002, 1), frequency = 12)
err2_lasso     <- ts(cumsum((err_lasso)^2), start = c(2002, 1), frequency = 12)
err2_constant  <- sum(err_benchmark^2)
# plot
plot.ts(err2_benchmark/err2_constant, lwd = 2,
        ylab = "Cumulated squared errors",
        main = "Forecast errors over time",
        xlab = "")
lines(err2_pcr/err2_constant, col = "steelblue", lwd = 2)
lines(err2_lasso/err2_constant, col = "darkred", lwd = 2)
rect(2008,-1,2009.5,1.5,col = rgb(0.6,0.1,0.1,1/4))
legend("topleft", c("Sample mean", "PCR", "LASSO"),
       col=c("black", "steelblue", "darkred"), lwd=6)
```