目录

Preface

This course was offered in 2015, as a response to postgraduate students' repeatedly request for training in coding skills. Programming is an essential skill for researchers in economic analysis. In the time of big data, empirical economic research becomes main stream.

I believe the best way to learn programming is to follow examples. These notes contain executable examples that illustrate R usage and econometric computational ideas.

Econometrics is interdisciplinary study involving economics, statistics, operational research, and computational science. No matter how beautiful is the theory, it will be of little use if it cannot be implemented. Thus econometric procedure must be accompanied with computational code. Given that many existing methods are encapsulated in canned packages, they do not cover new methods, which must be developed by econometrics who propose these procedures.

Our econometric procedures are difficult to commercialize. To facilitate spread, one way is to offer to code for free. The world runs on open-source software. Many open source software is indispensable for our daily work, for example, Linux, R, Python and LaTeX.

These lecture notes are products of an ongoing writing project. They are not intended to be comprehensive. I don't want to reinvent wheels. I refer to the relevant papers and surveys when there are excellent writings for mature topics.

Personal Reflection

Thirty years ago aspiring young econometricians picked up GAUSS. Twenty years ago the new blood began with MATLAB. R raised ten years ago when the time came to my generation. I have been using R since I started my postgraduate study in Peking University in 2005. R helps me with my daily research and teaching.

There are other candidates in statistical programming, for example Matlab, Python, Julia and Fortran. Each language has its own pros and cons. R has many advantages. First, it inherits the standard program syntax from C. It is quick to learn for anyone with prior programming experience. Moreover, once you master R, it is easy to switch to other language, if not R, in your workplace in the future.

Second, backed by a vast statistician community, R enjoys a large selection of packages, including the most recent ones. When they publish a paper, often times statisticians write and upload a companion R package to facilitate user adoption.

Third, R is free. It was the primary reason that I chose it at the very beginning. In the era of cloud computing, an algorithm written in R is easier to share, test, and improve. Anyone can run R code on any platform, free of license and fee headache.

R is not without limitations. For example, speed is a concern when running big and complex jobs. However, it will not be an issue in the problems that we will encounter in the first-year postgraduate study.

Lastly, learning a language is a non-trivial investment of our precious time. It is much more important to master one language with fluency than to know several languages.

R is not the only language available for computing. Why not Python? Python is a general purpose language, not a scientific computing language. We need to import external modules even for basic numerical operations. For example, I personally don't like `np.mean`, `np.var` and `np.log`, and its index rule starting from 0. For basic matrix manipulation, the default behavior of `numpy` is different from R.

Why not Julia? Julia is too young to have a big community. We would wait until it grows into more stable status. Moreover, the speed advantage does not help much in interactive usage in empirical research.

Over the years, I have had a taste of both Python and Julia. In my opinion, R so far best suits our purpose of learning a computing language for statistics and econometric analysis.

I was an expert in MATLAB, which is proprietary. I wrote those first few papers during and after my Ph.D study. However, I was never a fan of MATLAB's syntax, what was worse was its ugly functions. MATLAB may still linger in some areas in engineering, but it is a dinosaur fossil buried under the wonderland of big data.

Prerequisite

For this course, please install R or Microsoft Open R. An integrated development environment (IDE) is also highly desirable. It makes programming user-friendly and enjoyable. RStudio is recommended.

Structure

The book version can be partitioned into three parts: R, Econometrics, and Machine Learning.

The first two lectures cover basic R and advanced R. Advanced R taught skills about speeding up matrix manipulation, parallel computing and remote computing. The following two lectures cover simulation exercises and simulation-based methods in econometrics. The last two lectures talk about machine learning methods. Machine learning is relatively new for economists. Most economics programs only introduce a few algorithms but do not cover the theoretical components. We try to provide a review starting from the conventional nonparametric statistical methods.

Packages

```
install.packages(c("plyr", "foreach", "doParallel",
                   "sampleSelection", "AER", "mcmc",
                   "randomForest", "gbm"))
```

1 Basic R

1.1 Introduction

One cannot acquire a new programming language without investing numerous hours. R-Introduction is an official manual maintained by the R core team. It was the first document that I perused painstakingly when I began to learn R in 2005. After so many years, this is still the best starting point for you to have a taste.

This lecture quickly sketches some key points of the manual, while you should carefully go over R-Introduction after today's lecture.

1.2 Help System

The help system is the first thing we must learn for a new language. In R, if we know the exact name of a function and want to check its usage, we can either call `help(function_name)` or a single question mark `?function_name`. If we do not know the exact function name, we can instead use the double question mark `??key_words`. It will provide a list of related function names from a fuzzy search.

Example: `?seq`, `??sequence`

1.3 Vector

A *vector* is a collection of elements of the same type, say, integer, logical value, real number, complex number, characters or factor.R does not require explicit type declaration.

`<-` assigns the value on its right-hand side to a self-defined variable name on its left-hand side. `=` is an alternative for assignment.

`c()` combines two or more vectors into a long vector.

Binary arithmetic operations `+`, `-`, `*` and `/` are performed element by element by default. So are the binary logical operations `& | !=`.

Factor is a categorical number. *Character* is text.

Missing values in R is represented as `NA` (Not Available). When some operations are not allowed, say, `log(-1)`, R returns `NaN` (Not a Number).

Vector selection is specified in square bracket `a[ ]` by either positive integer or logical vector. The index initiates from 1, not 0 (Python's rule).

Example

Logical vector operation.

```
# logical vectors
logi_1 <- c(T, T, F)
logi_2 <- c(F, T, T)

logi_12 <- logi_1 & logi_2
print(logi_12)
```

```
## [1] FALSE  TRUE FALSE
```

1.4 Array and Matrix

An array is a table of numbers.
A matrix is a 2-dimensional array.

- array arithmetic: element-by-element. Caution must be exercised in binary operations involving two objects of different length. This is error-prone.
- `%*%`, `solve`, `eigen`

Example

OLS estimation with one x regressor and a constant. Graduate textbook expresses the OLS in matrix form

$$\hat{\beta} = (X'X)^{-1}X'y.$$

To conduct OLS estimation in R, we literally translate the mathematical expression into code.

Step 1: We need data Y and X to run OLS. We simulate an artificial dataset.

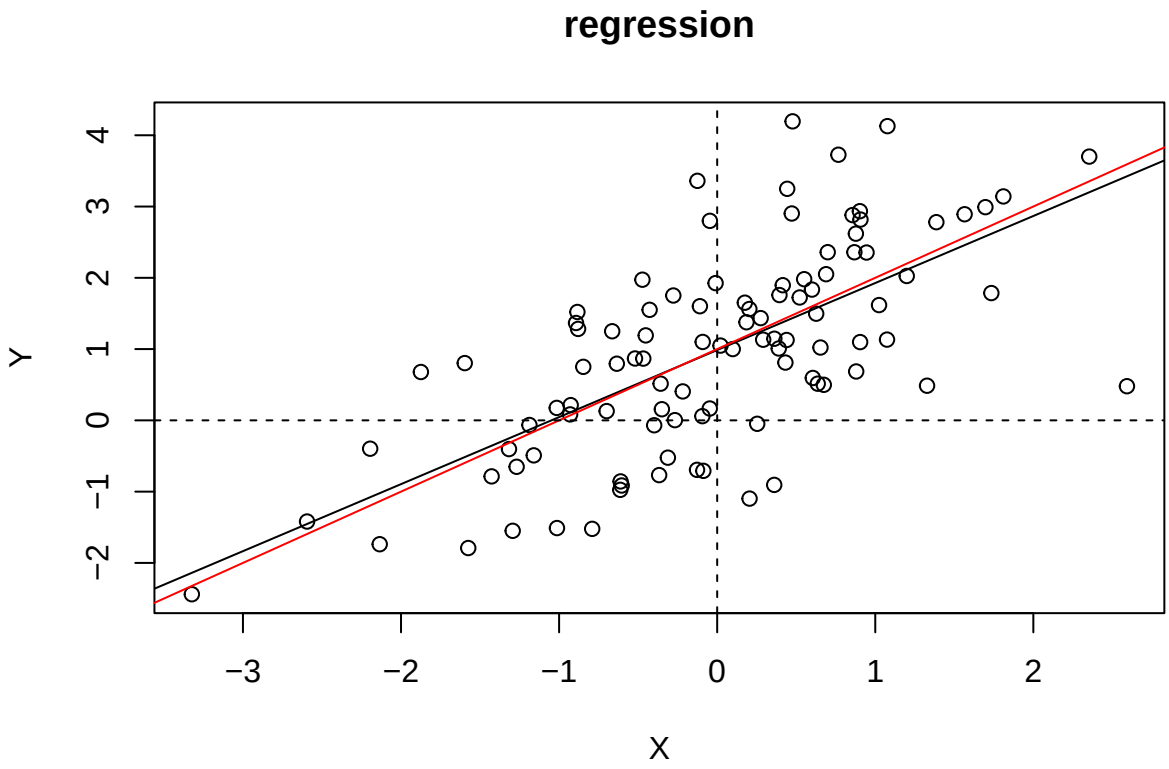
```
# simulate data
rm(list = ls())
set.seed(111) # can be removed to allow the result to change

# set the parameters
n <- 100
b0 <- matrix(1, nrow = 2)

# generate the data
e <- rnorm(n)
X <- cbind(1, rnorm(n))
Y <- X %*% b0 + e
```

Step 2: translate the formula to code

```
# OLS estimation
bhat <- solve(t(X) %*% X, t(X) %*% Y)
```

Step 3 (additional): plot the regression graph with the scatter points and the regression line. Further compare the regression line (black) with the true coefficient line (red).

```
# plot
plot(y = Y, x = X[, 2], xlab = "X", ylab = "Y", main = "regression")
abline(a = bhat[1], b = bhat[2])
abline(a = b0[1], b = b0[2], col = "red")
abline(h = 0, lty = 2)
abline(v = 0, lty = 2)
```



Step 4: In econometrics we are often interested in hypothesis testing. The t-statistic is widely used. To test the null $H_0 : \beta_2 = 1$, we compute the associated t-statistic. Again, this is a translation.

$$t = \frac{\hat{\beta}_2 - \beta_{02}}{\hat{\sigma}_{\hat{\beta}_2}} = \frac{\hat{\beta}_2 - \beta_{02}}{\sqrt{[(X'X)^{-1}\hat{\sigma}^2]_{22}}}.$$

where $[\cdot]_{22}$ is the (2,2)-element of a matrix.

```
# calculate the t-value
bhat2 <- bhat[2] # the parameter we want to test
e_hat <- Y - X %*% bhat
sigma_hat_square <- sum(e_hat^2) / (n - 2)
Sigma_B <- solve(t(X) %*% X) * sigma_hat_square
t_value_2 <- (bhat2 - b0[2]) / sqrt(Sigma_B[2, 2])
cat("The t-statistic =", t_value_2)
```

```
## The t-statistic = -0.5615293
```

1.5 Package

A pure clean installation of R is small, but R has an extensive ecosystem of add-on packages. This is the unique treasure for R users, and other languages like Python or MATLAB are not even close. Most packages are hosted on CRAN. A common practice today is that statisticians upload a

package to CRAN after they write or publish a paper with a new statistical method. They promote their work via CRAN, and users have easy access to the state-of-the-art methods.

A package can be installed by `install.packages("package_name")`. To invoke a function of a package, we can either call the function name after import the package by `library(package_name)` into the current session, or use `package_name::function_name`. The former imports all functions in the package, and sometimes can cause conflict with other functions of the same name. The latter method is preferred to keep the environment clean.

Applied Econometrics with R by Christian Kleiber and Achim Zeileis is a useful book. It also has a companion package `AER` that contains popular econometric methods such as instrumental variable regression and robust variance.

Before we can "knit" in R-studio the Rmd file to produce the pdf document you are reading at this moment, we have to install several packages such as knitr and those it depends on.

1.6 Mixed Data Types

A vector only contains one type of elements. *list* is a basket for objects of various types. It can serve as a container when a procedure returns more than one useful object. For example, when we invoke `eigen`, we are interested in both eigenvalues and eigenvectors, which are stored into `$value` and `$vector`, respectively.

data.frame is a two-dimensional table that stores the data, similar to a spreadsheet in Excel. A matrix is also a two-dimensional table, but it only accommodates one type of elements. Real world data can be a collection of integers, real numbers, characters, categorical numbers and so on. Data frame is the default way to organize data of mixed type in R. `tibble` is a new and refined alternative data frame type.

Example

This is a data set in a graduate-level econometrics textbook. We load the data into memory and display the first 6 records.

```
library(AER)
data("CreditCard")
head(CreditCard)
```

```
##   card reports      age income       share expenditure owner selfemp dependents
## 1  yes       0 37.66667 4.5200 0.033269910  124.983300   yes      no          3
## 2  yes       0 33.25000 2.4200 0.005216942    9.854167    no      no          3
## 3  yes       0 33.66667 4.5000 0.004155556   15.000000   yes      no          4
## 4  yes       0 30.50000 2.5400 0.065213780  137.869200    no      no          0
## 5  yes       0 32.16667 9.7867 0.067050590  546.503300   yes      no          2
## 6  yes       0 23.25000 2.5000 0.044438400   91.996670    no      no          0
##   months majorcards active
## 1     54          1     12
## 2     34          1     13
## 3     58          1      5
```

```
## 4     25          1      7
## 5     64          1      5
## 6     54          1      1
```

```
head(tibble::as_tibble(CreditCard))
```

```
## # A tibble: 6 x 12
##   card  reports   age income   share expenditure owner selfemp dependents months
##   <fct>   <dbl> <dbl>  <dbl>   <dbl>       <dbl> <fct> <fct>        <dbl>  <dbl>
## 1 yes         0  37.7   4.52 0.0333       125.   yes   no               3     54
## 2 yes         0  33.2   2.42 0.00522        9.85 no    no               3     34
## 3 yes         0  33.7   4.5  0.00416       15    yes   no               4     58
## 4 yes         0  30.5   2.54 0.0652       138.   no    no               0     25
## 5 yes         0  32.2   9.79 0.0671       547.   yes   no               2     64
## 6 yes         0  23.2   2.5  0.0444        92.0  no    no               0     54
## # ... with 2 more variables: majorcards <dbl>, active <dbl>
```

1.7 Input and Output

Raw data is often saved in ASCII file or Excel. I discourage the use of Excel spreadsheet in data analysis, because the underlying structure of an Excel file is too complicated for statistical software to read. I recommend the use of `csv` format, a plain ASCII file format.

`read.table()` or `read.csv()` imports data from an ASCII file into an R session. `write.table()` or `write.csv()` exports the data in an R session to an ASCII file.

Example

Besides loading a data file on the local hard disk, We can directly download data from internet. Here we show how to retrieve the stock daily data of *Apple Inc.* from *Yahoo Finance*, and save the dataset locally. A package called `quantmod` is used.

```
quantmod::getSymbols("AAPL", src = "yahoo")
```

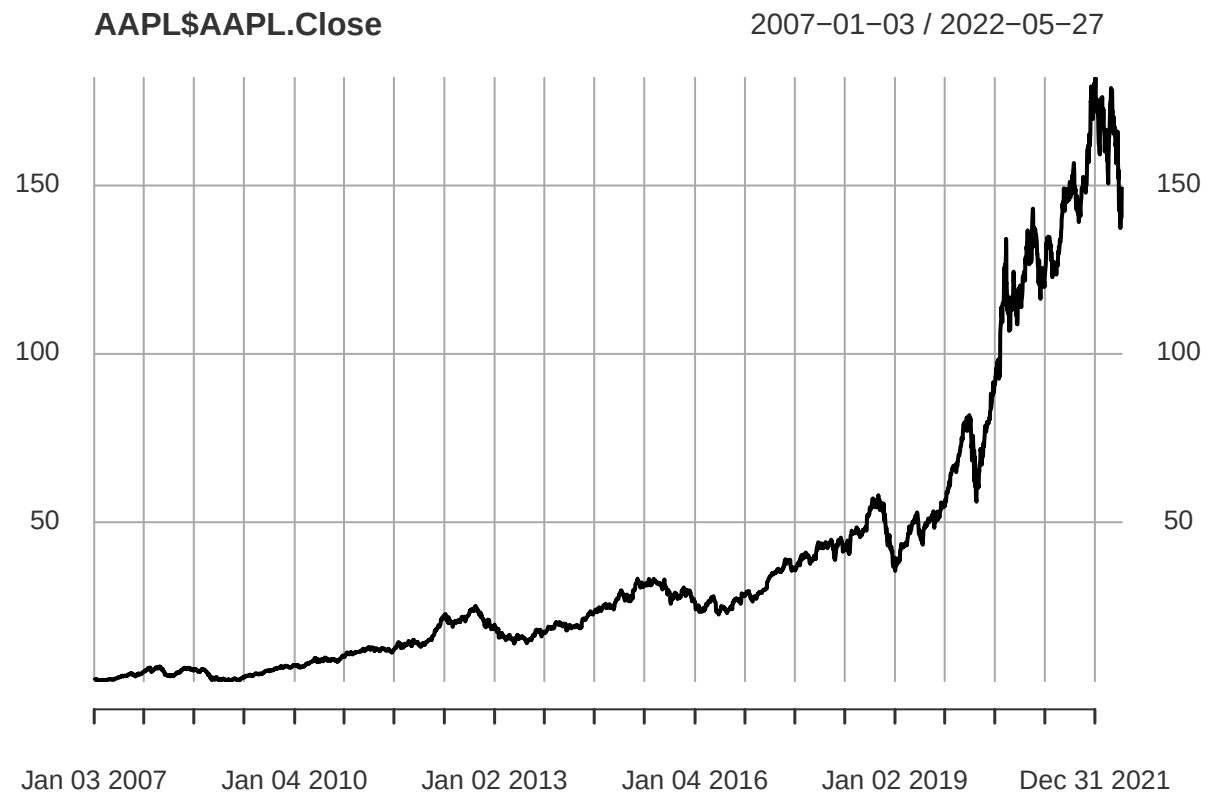
```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

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

```
tail(AAPL)
```

```
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
## 2022-05-20    139.09    140.70   132.61     137.59   137194600        137.59
## 2022-05-23    137.79    143.26   137.65     143.11   117726300        143.11
## 2022-05-24    140.81    141.97   137.33     140.36   104132700        140.36
## 2022-05-25    138.43    141.79   138.34     140.52    92482700        140.52
## 2022-05-26    137.39    144.34   137.14     143.78    90601500        143.78
## 2022-05-27    145.39    149.68   145.26     149.64    90796900        149.64
```

```
plot(AAPL$AAPL.Close)
```

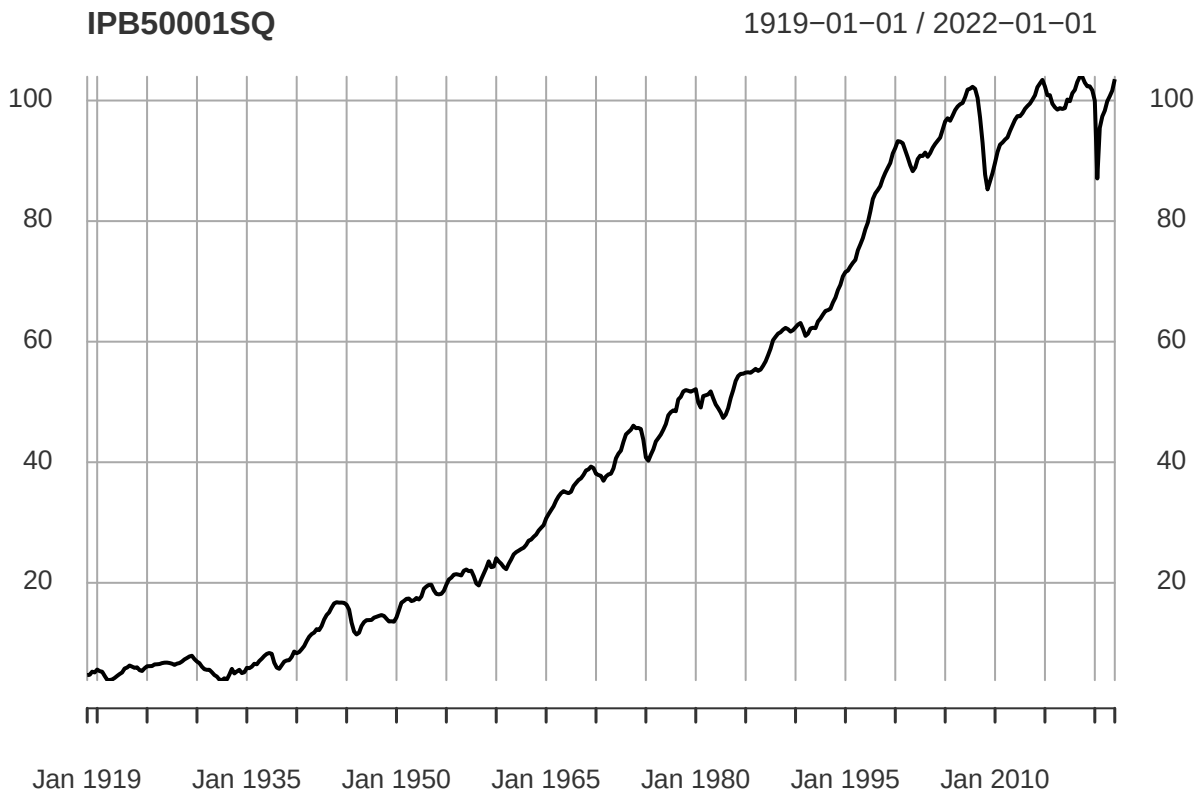


Another example: Quarterly US Industrial Production Index

```
quantmod::getSymbols.FRED(Symbols = c("IPB50001SQ"), env = .GlobalEnv)
```

```
## [1] "IPB50001SQ"
```

```
plot(IPB50001SQ)
```



1.8 Statistics

R is a language created by statisticians. It has elegant built-in statistical functions. `p` (probability), `d` (density for a continuous random variable, or mass for a discrete random variable), `q` (quantile), `r` (random variable generator) are used ahead of the name of a probability distribution, such as `norm` (normal), `chisq` (χ^2), `t` (t), `weibull` (Weibull), `cauchy` (Cauchy), `binomial` (binomial), `pois` (Poisson), to name a few.

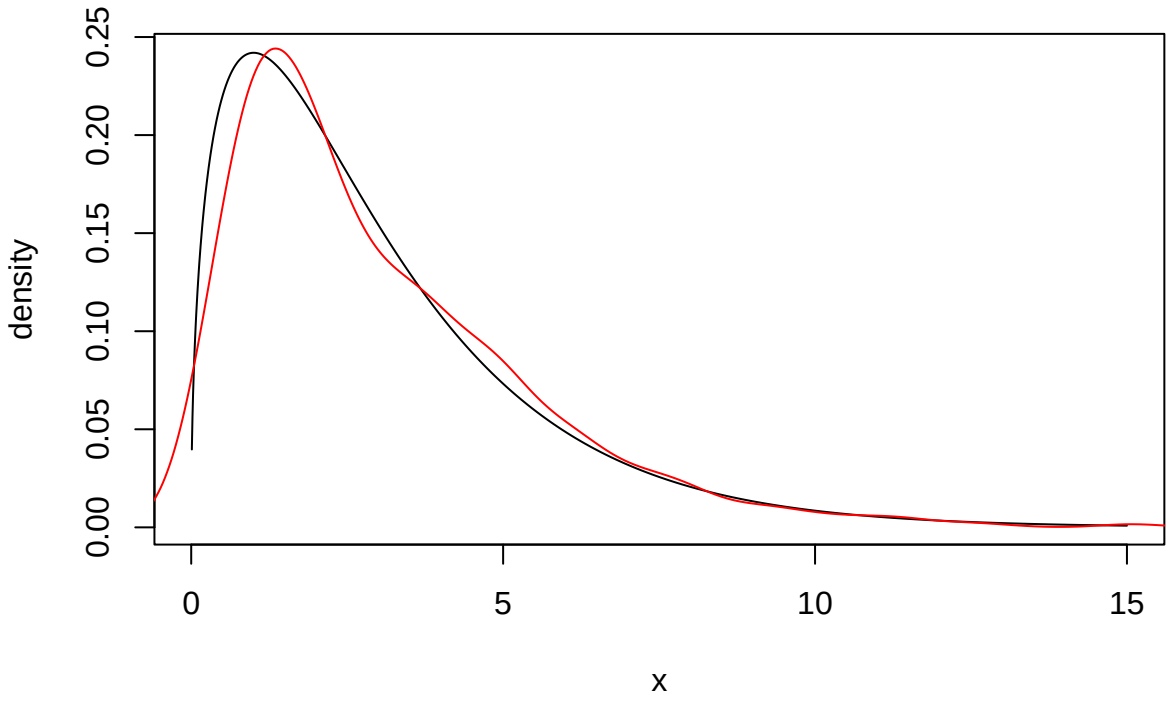
Example

This example illustrates the sampling error.

1. Plot the density of $\chi^2(3)$ over an equally spaced grid system `x_axis = seq(0.01, 15, by = 0.01)` (black line).
2. Generate 1000 observations from $\chi^2(3)$ distribution. Plot the kernel density, a nonparametric estimation of the density (red line).
3. Calculate the 95th quantile and the empirical probability of observing a value greater than the 95-th quantile. In population, this value should be 5%. What is the number in this experiment?

```
set.seed(888)
x_axis <- seq(0.01, 15, by = 0.01)
```

```
y <- dchisq(x_axis, df = 3)
plot(y = y, x = x_axis, type = "l", xlab = "x", ylab = "density")
z <- rchisq(1000, df = 3)
lines(density(z), col = "red")
```




```
crit <- qchisq(.95, df = 3)

mean(z > crit)
```

```
## [1] 0.047
```

1.9 User-defined Function

R has numerous built-in functions. However, in practice we will almost always have some DIY functionality to be used repeatedly. It is highly recommended to encapsulate it into a user-defined function. There are important advantages:

1. In the developing stage, it allows us to focus on a small chunk of code. It cuts an overwhelmingly big project into manageable pieces.
2. A long script can have hundreds or thousands of variables. Variables defined inside a function are local. They will not be mixed up with those outside of a function. Only the input and the output of a function have interaction with the outside world.

3. If a revision is necessary, We just need to change one place. We don't have to repeat the work in every place where it is invoked.

The format of a user-defined function in R is

```
function_name <- function(input) {
  expressions
  return(output)
}
```

Example

If the central limit theorem is applicable, then we can calculate the 95% two-sided asymptotic confidence interval as

$$\left(\hat{\mu} - \frac{1.96}{\sqrt{n}}\hat{\sigma}, \hat{\mu} + \frac{1.96}{\sqrt{n}}\hat{\sigma}\right)$$

from a given sample. It is an easy job, but I am not aware there is a built-in function in R to do this.

```
# construct confidence interval

CI <- function(x) {
  # x is a vector of random variables

  n <- length(x)
  mu <- mean(x)
  sig <- sd(x)
  upper <- mu + 1.96 / sqrt(n) * sig
  lower <- mu - 1.96 / sqrt(n) * sig
  return(list(lower = lower, upper = upper))
}
```

1.10 Flow Control

Flow control is common in all programming languages. `if` is used for choice, and `for` or `while` is used for loops.

Example

Calculate the empirical coverage probability of a Poisson distribution of degrees of freedom 2. We conduct this experiment for 1000 times.

```
Rep <- 1000
sample_size <- 100
capture <- rep(0, Rep)

for (i in 1:Rep) {
  mu <- 2
  x <- rpois(sample_size, mu)
  bounds <- CI(x)
```

```
  capture[i] <- ((bounds$lower <= mu) & (mu <= bounds$upper))
}
cat("the emprical size = ", mean(capture)) # empirical size
```

```
## the emprical size =  0.938
```

1.11 Statistical Model

Statistical models are formulated as `y~x`, where `y` on the left-hand side is the dependent variable, and `x` on the right-hand side is the explanatory variable. The built-in OLS function is `lm`. It is called by `lm(y~x, data = data_frame)`.

All built-in regression functions in R share the same structure. Once one type of regression is understood, it is easy to extend to other regressions.

1.11.1 A Linear Regression Example

This is a toy example with simulated data.

```
T <- 100
p <- 1

b0 <- 1
# Generate data
x <- matrix(rnorm(T * p), T, 1)
y <- x %*% b0 + rnorm(T)

# Linear Model
result <- lm(y ~ x)
summary(result)
```

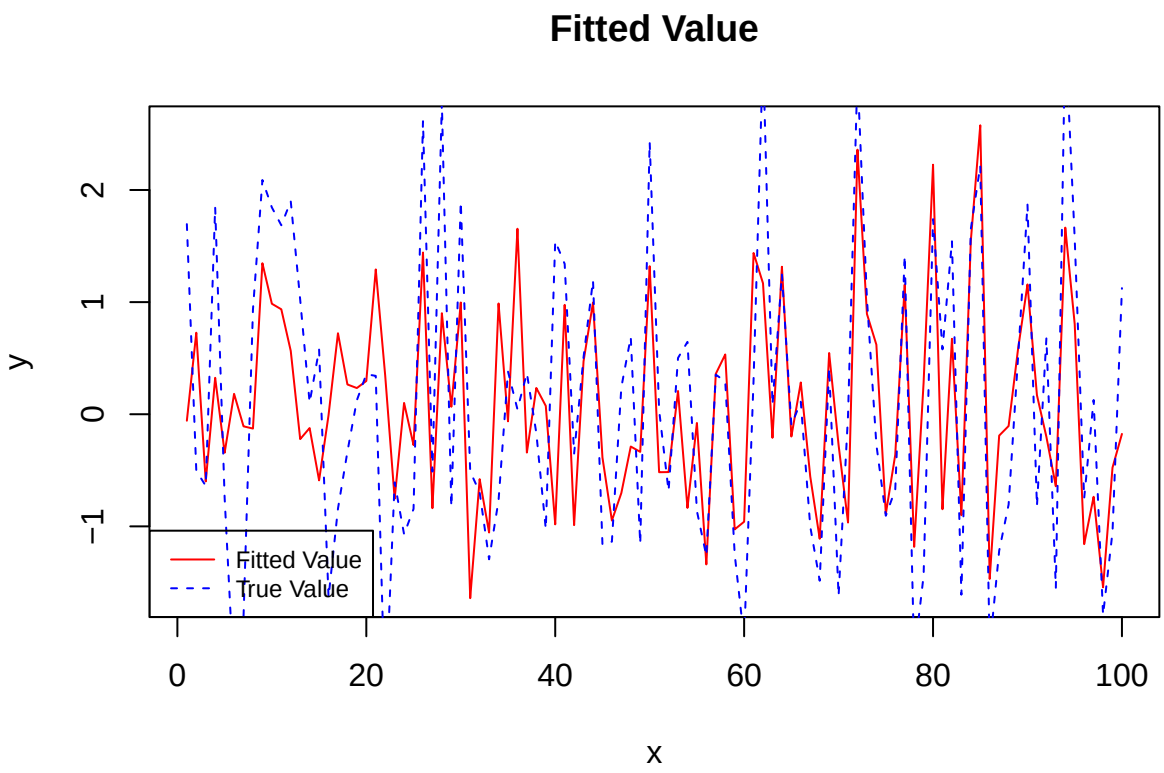
```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.96282 -0.70247 -0.02817  0.75194  2.52548
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.04445    0.10221   0.435    0.665
## x            0.84427    0.09604   8.791 5.06e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1.021 on 98 degrees of freedom
## Multiple R-squared:  0.4409, Adjusted R-squared:  0.4352
## F-statistic: 77.29 on 1 and 98 DF,  p-value: 5.06e-14
```

The `result` object is a list containing the regression results. As shown in the results, we can easily read the estimated coefficients, t-test results, F-test results, and the R-squared.

We can plot the true value of y and fitted value to examine whether the regression model fit the data well.

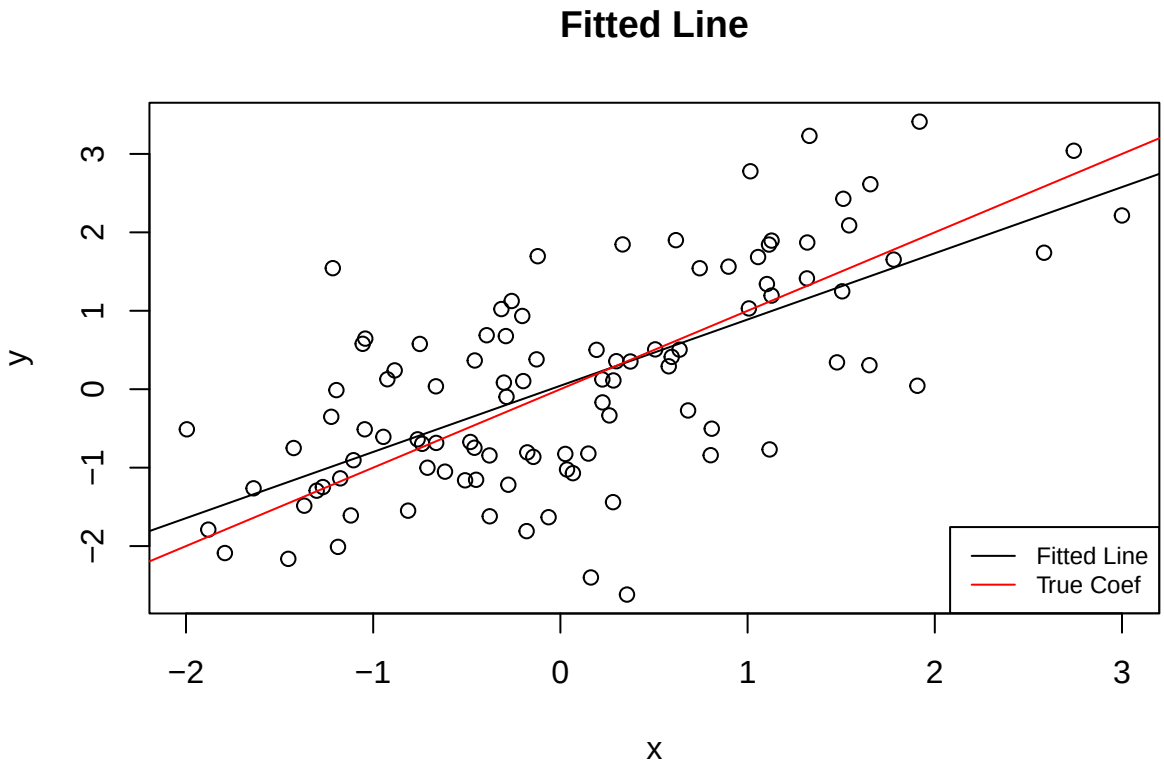
```
plot(result$fitted.values,
  col = "red", type = "l", xlab = "x", ylab = "y",
  main = "Fitted Value"
)
lines(y, col = "blue", type = "l", lty = 2)
legend("bottomleft",
  legend = c("Fitted Value", "True Value"),
  col = c("red", "blue"), lty = 1:2, cex = 0.75
)
```



Then we plot the best fitted line.

```
plot(y = y, x = x, xlab = "x", ylab = "y", main = "Fitted Line")
abline(a = result$coefficients[1], b = result$coefficients[2])
abline(a = 0, b = b0, col = "red")
```

```
legend("bottomright",
  legend = c("Fitted Line", "True Coef"),
  col = c("black", "red"), lty = c(1, 1), cex = 0.75
)
```



Here we give another example about the relationship between the height and weight of women. The women dataset is from the package `datasets`, which is a built-in package shipped with R installation. This package contains a variety of datasets. For a complete list, use `library(help = "datasets")`

```
# univariate
reg1 <- lm(height ~ weight, data = women)

# multivariate
reg2 <- lm(height ~ weight + I(weight^2), data = women)
# "weight^2" is a square term.
# "I()" is used to inhibit the formula operator "+"
# from being interpreted as an arithmetical one.

summary(reg1)
```

```
##
## Call:
```

```
## lm(formula = height ~ weight, data = women)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.83233 -0.26249  0.08314  0.34353  0.49790
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.723456   1.043746   24.64 2.68e-12 ***
## weight       0.287249   0.007588   37.85 1.09e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.44 on 13 degrees of freedom
## Multiple R-squared:  0.991,  Adjusted R-squared:  0.9903
## F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

```
summary(reg2)
```

```
##
## Call:
## lm(formula = height ~ weight + I(weight^2), data = women)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.105338 -0.035764 -0.004898  0.049430  0.141593
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.175e+01  1.720e+00   -6.83 1.82e-05 ***
## weight       8.343e-01  2.502e-02   33.35 3.36e-13 ***
## I(weight^2) -1.973e-03  9.014e-05  -21.89 4.84e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07158 on 12 degrees of freedom
## Multiple R-squared:  0.9998, Adjusted R-squared:  0.9997
## F-statistic: 2.732e+04 on 2 and 12 DF,  p-value: < 2.2e-16
```

1.12 Reading

Wickham and Grolemund: Ch 1, 2, 4, 8, 19 and 20

2 Advanced R

2.1 Introduction

In this lecture, we will talk about efficient computation in R.

- R is a vector-oriented language. In most cases, vectorization speeds up computation.
- We turn to more CPUs for parallel execution to save time if there is no more room to optimize the code to improve the speed.
- Servers are accessed remotely. Communicating with a remote cluster is different from operating a local machine.

2.2 Vectorization

Despite mathematical equivalence, various ways of calculation can perform distinctively in terms of computational speed.

Does computational speed matter? For a job that takes less than a minutes, the time difference is not a big deal. But sometimes economic problems can be clumsy. For structural estimation commonly seen in industrial organization, a single estimation can take up to a week. In econometrics, other computational intensive procedures include bootstrap, simulated maximum likelihood and simulated method of moments. Even if a single execution does not take much time, repeating such a procedure for thousands of replications will consume a non-trivial duration. Moreover, machine learning methods that crunch big data usually involve tuning parameters, so the same procedure must be carried out at each point of a grid of tuning parameters. For example, the preferred algorithm in Lin et al. (2020) takes 8 hours on a 24-core remote server to find out the best combination of tuning parameters. For those problems, code optimization is essential.

Of course, optimizing code takes human time. It is a balance of human time and machine time.

Example

In OLS regression, under homoskedasticity

$$\sqrt{n}\left(\widehat{\beta}-\beta_0\right) \xrightarrow{d} N\left(0, \sigma^2\left(E\left[x_i x_i'\right]\right)^{-1}\right)$$

where the asymptotic variance can be consistently estimated by $(X'X)^{-1}\sum_{i=1}^{n}\widehat{e}^2$. However, under heteroskedasticity

$$\sqrt{n}\left(\widehat{\beta}-\beta_0\right) \xrightarrow{d} N\left(0, E\left[x_i x_i'\right]^{-1}\operatorname{var}\left(x_i e_i\right)E\left[x_i x_i'\right]^{-1}\right)$$

where $\operatorname{var}(x_i e_i)$ can be estimated by

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} x_i x_i'\widehat{e}_i^2}_{\text{opt 1}} = \underbrace{\frac{1}{n}X'DX}_{\text{opt 2 and 3}} = \underbrace{\frac{1}{n}\left(X'D^{1/2}\right)\left(D^{1/2}X\right)}_{\text{opt 4}}$$

where D is a diagonal matrix of $\left(\widehat{\epsilon}_1^2, \widehat{\epsilon}_{2,}^2, \ldots, \widehat{\epsilon}_n^2\right)$. There are at least 4 mathematically equivalent ways to compute the "meat" of the sandwich form.

1. literally sum $\hat{e}_i^2 x_i x_i'$ over $i = 1, \ldots, n$ one by one.
2. $X'\text{diag}(\hat{e}^2)X$, with a dense central matrix.
3. $X'\text{diag}(\hat{e}^2)X$, with a sparse central matrix.
4. Do cross product to `X*e_hat`. It takes advantage of the element-by-element operation in R.

We first generate the data of binary response and regressors. Due to the discrete nature of the dependent variable, the error term in the linear probability model is heteroskedastic. It is necessary to use the heteroskedastic-robust variance to consistently estimate the asymptotic variance of the OLS estimator. The code chunk below estimates the coefficients and obtains the residual.

```
# an example of robust variance matrix.
# compare the implementation via matrix, Matrix (package) and vecteroization.

# n = 5000; Rep = 10; # Matrix is quick, matrix is slow, adding is OK

source("data_example/lec2.R")

n <- 50
Rep <- 1000

data.Xe <- lpm(n) # see the function in "data_example/lec2.R"
X <- data.Xe$X
e_hat <- data.Xe$e_hat

XXe2 <- matrix(0, nrow = 2, ncol = 2)
```

We run the 4 estimators for the same data, and compare the time.

```
for (opt in 1:4) {
  pts0 <- Sys.time()

  for (iter in 1:Rep) {
    set.seed(iter) # to make sure that the data used
    # different estimation methods are the same

    if (opt == 1) {
      for (i in 1:n) {
        XXe2 <- XXe2 + e_hat[i]^2 * X[i, ] %*% t(X[i, ])
      }
    } else if (opt == 2) { # the vectorized version with dense matrix
      e_hat2_M <- matrix(0, nrow = n, ncol = n)
      diag(e_hat2_M) <- e_hat^2
      XXe2 <- t(X) %*% e_hat2_M %*% X
    } else if (opt == 3) { # the vectorized version with sparse matrix
      e_hat2_M <- Matrix::Matrix(0, ncol = n, nrow = n)
      diag(e_hat2_M) <- e_hat^2
```

```
      XXe2 <- t(X) %*% e_hat2_M %*% X
    } else if (opt == 4) { # the best vectorization method. No waste
      Xe <- X * e_hat
      XXe2 <- t(Xe) %*% Xe
    }

    XX_inv <- solve(t(X) %*% X)
    sig_B <- XX_inv %*% XXe2 %*% XX_inv
  }
  cat("n =", n, ", Rep =", Rep, ", opt =", opt, ", time =", Sys.time() - pts0, "\n")
}
```

```
## n = 50 , Rep = 1000 , opt = 1 , time = 0.377593
## n = 50 , Rep = 1000 , opt = 2 , time = 0.116869
## n = 50 , Rep = 1000 , opt = 3 , time = 2.004598
## n = 50 , Rep = 1000 , opt = 4 , time = 0.04244399
```

We clearly see the difference in running time, though the 4 methods are mathematically the same. When n is small, `matrix` is fast and `Matrix` is slow; the vectorized version is the fastest. When n is big, `matrix` is slow and `Matrix` is fast; the vectorized version is still the fastest.

In this simulation exercise, we repeat the procedure many times to make the time comparison more evident, for a single execution takes very short time in this simple operation. A real-data example is in `data_example/IPUMS.R` with 234 thousand observations, where the time difference is dramatic but the intuitive solution indeed does not take much time. It demonstrates the usefulness of vectorization. Vectorization can significantly saves computing time in more complicated operations, for example, in heteroskedastic and autocorrelation consistent variance estimation (HAC) in econometrics which involves many layers of matrices.

2.3 Efficient Loop

R was the heir of S, an old language. R evolves with packages that are designed to adapt to new big data environment. Many examples can be found in Wickham and Grolemund (2016). Here we introduce `plyr`.

In standard `for` loops, we have to do a lot of housekeeping work. Hadley Wickham's `plyr` simplifies the job and facilitates parallelization.

Example

Here we calculate the empirical coverage probability of a Poisson distribution of degrees of freedom 2. We first write a user-defined function `CI` for confidence interval, which was used in the last lecture.

This is a standard `for` loop.

```
Rep <- 100000
sample_size <- 1000
mu <- 2
```

```
source("data_example/lec2.R")
# append a new outcome after each loop
pts0 <- Sys.time() # check time
for (i in 1:Rep) {
  x <- rpois(sample_size, mu)
  bounds <- CI(x)
  out_i <- ((bounds$lower <= mu) & (mu <= bounds$upper))
  if (i == 1) {
    out <- out_i
  } else {
    out <- c(out, out_i)
  }
}

pts1 <- Sys.time() - pts0 # check time elapse
cat("loop without pre-definition takes", pts1, "seconds\n")
```

```
## loop without pre-definition takes 24.6467 seconds
```

```
# pre-define a container
out <- rep(0, Rep)
pts0 <- Sys.time() # check time
for (i in 1:Rep) {
  x <- rpois(sample_size, mu)
  bounds <- CI(x)
  out[i] <- ((bounds$lower <= mu) & (mu <= bounds$upper))
}

pts1 <- Sys.time() - pts0 # check time elapse
cat("loop with pre-definition takes", pts1, "seconds\n")
```

```
## loop with pre-definition takes 10.45834 seconds
```

Pay attention to the line `out = rep(0, Rep)`. It *pre-defines* a vector `out` to be filled by `out[i] = ( ( bounds$lower <= mu  ) & (mu <= bounds$upper) )`. The computer opens a continuous patch of memory for the vector `out`. When new result comes in, the old element is replaced. If we do not pre-define `out` but append one more element in each loop, the length of `out` will change in each replication and every time a new patch of memory will be assigned to store it. The latter approach will spend much more time just to locate the vector in the memory.

`out` is the result container. In a `for` loop, we pre-define a container, and replace the elements of the container in each loop by explicitly calling the index.

In contrast, a `plyr` loop saves the house keeping chores, and makes it easier to parallelize. In the example below, we encapsulate the chunk in the `for` loop as a new function `capture`, and run the replication via `__ply`. `__ply` is a family of functions. `ldply` here means that the input is a list (`l`) and the output is a data frame (`d`) .

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 4.1.3
```

```
capture <- function(i) {
  x <- rpois(sample_size, mu)
  bounds <- CI(x)
  return((bounds$lower <= mu) & (mu <= bounds$upper))
}

pts0 <- Sys.time() # check time
out <- ldply(.data = 1:Rep, .fun = capture)

pts1 <- Sys.time() - pts0 # check time elapse
cat("plyr loop takes", pts1, "seconds\n")
```

```
## plyr loop takes 9.442748 seconds
```

This example is so simple that the advantage of `plyr` is not dramatic. The difference in coding will be noticeable in complex problems with big data frames. In terms of speed, `plyr` does not run much faster than a `for` loop. They are of similar performance. Parallel computing will be our next topic. It is quite easy to implement parallel execution with `plyr`—we just need to change one argument in the function.

2.4 Parallel Computing

Parallel computing becomes essential when the data size is beyond the storage of a single computer, for example Q. Li et al. (2018). Here we explore the speed gain of parallel computing on a multicore machine.

Here we introduce how to coordinate multiple cores on a single computer. The packages `foreach` and `doParallel` are useful for parallel computing. Below is the basic structure. `registerDoParallel(number)` prepares a few CPU cores to accept incoming jobs.

```
library(plyr); library(foreach); library(doParallel)

registerDoParallel(a_number) # opens specified number of CPUs

out <- foreach(icount(Rep), .combine = option) %dopar% {
  my_expressions
}
```

If we have two CPUs running simultaneously, in theory we can cut the time to a half of that on a single CPU. Is that what happening in practice?

Example

Compare the speed of a parallel loop and a single-core sequential loop.

```
library(foreach)
```

```
## Warning: package 'foreach' was built under R version 4.1.3
```

```
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 4.1.3
```

```
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 4.1.3
```

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

```
registerDoParallel(2) # open 2 CPUs

pts0 <- Sys.time() # check time

out <- foreach(icount(Rep), .combine = c) %dopar% {
  capture()
}

pts1 <- Sys.time() - pts0 # check time elapse
cat("parallel loop takes", pts1, "seconds\n")
```

```
## parallel loop takes 40.69596 seconds
```

Surprisingly, the above code block of parallel computing runs even more slowly. It is because the task in each loop can be done in very short time. In contrast, the code chunk below will tell a different story. There the time in each loop is non-trivial, and then parallelism dominates the overhead of the CPU communication. The only difference between the two implementations below is that the first uses %dopar% and the latter uses %do%.

```
Rep <- 200
sample_size <- 2000000

registerDoParallel(8) # change the number of open CPUs according to
# the specification of your computer

pts0 <- Sys.time() # check time
out <- foreach(icount(Rep), .combine = c) %dopar% {
  capture()
}

cat("8-core parallel loop takes", Sys.time() - pts0 , "seconds\n")
```

```
## 8-core parallel loop takes 8.118207 seconds
```

```
pts0 <- Sys.time()
out <- foreach(icount(Rep), .combine = c) %do% {
  capture()
```

```
}

cat("single-core loop takes", Sys.time() - pts0 , "seconds\n")
```

```
## single-core loop takes 28.52989 seconds
```

2.5 Cloud Computing

Investing money from our own pocket to an extremely powerful laptop to conduct heavy-lifting computational work is unnecessary. (i) We do not run these long jobs every day, it is more cost efficient to share a workhorse. (ii) We cannot keep our laptop always on when we move it around. The right solution is cloud computing on a server.

Many of us have experience with cloud storage, such as Dropbox and Baidu Netdisk. Few people are exposed to cloud computing. However, no fundamental difference lies between local and cloud computing. We prepare in the cloud serve the data and code, open a shell for communication, run the code, and collect the results. One potential obstacle is dealing with a command-line-based operation system. Such command line tools is the norm of life two or three decades ago, but today we mostly work in a graphic operating system like Windows or OSX. For Windows users (I am one of them), I recommend `Git Bash` as a shell, and `WinSCP`, a graphic interface for input and output.

Cloud computing also provides a strong justification for open-source languages such as R or Python. These open-source languages can be installed on as many remote serves as the resource permits. In contrast, proprietary software will be prohibitively expensive for server licensing.

2.5.1 Command Line

Most servers run Unix/Linux operation system. Here are a few commands for basic operations.

- `mkdir`: make directory
- `cd`: change directory
- `copy`: copy files
- `top`: check login status
- `screen`: a separated screen for isolation
- `ssh`: user@address
- start a program

Our department's computation infrastructure has been improving. A server dedicated to professors is a 32-core machine. Students also have access to a powerful multi-core computer.

1. Log in `econsuper.econ.cuhk.edu.hk`;
2. Upload R scripts and data to the server;
3. In a shell, run `R --vanilla <file_name.R> result_file_name.out`;
4. To run a command in the background, add `&` at the end of the above command.

This example comes from Lin et al. (2020). As a demonstration, we only use 15% of the data and a sparse grid of tuning parameters. It makes about 9 minutes with 24 cores.

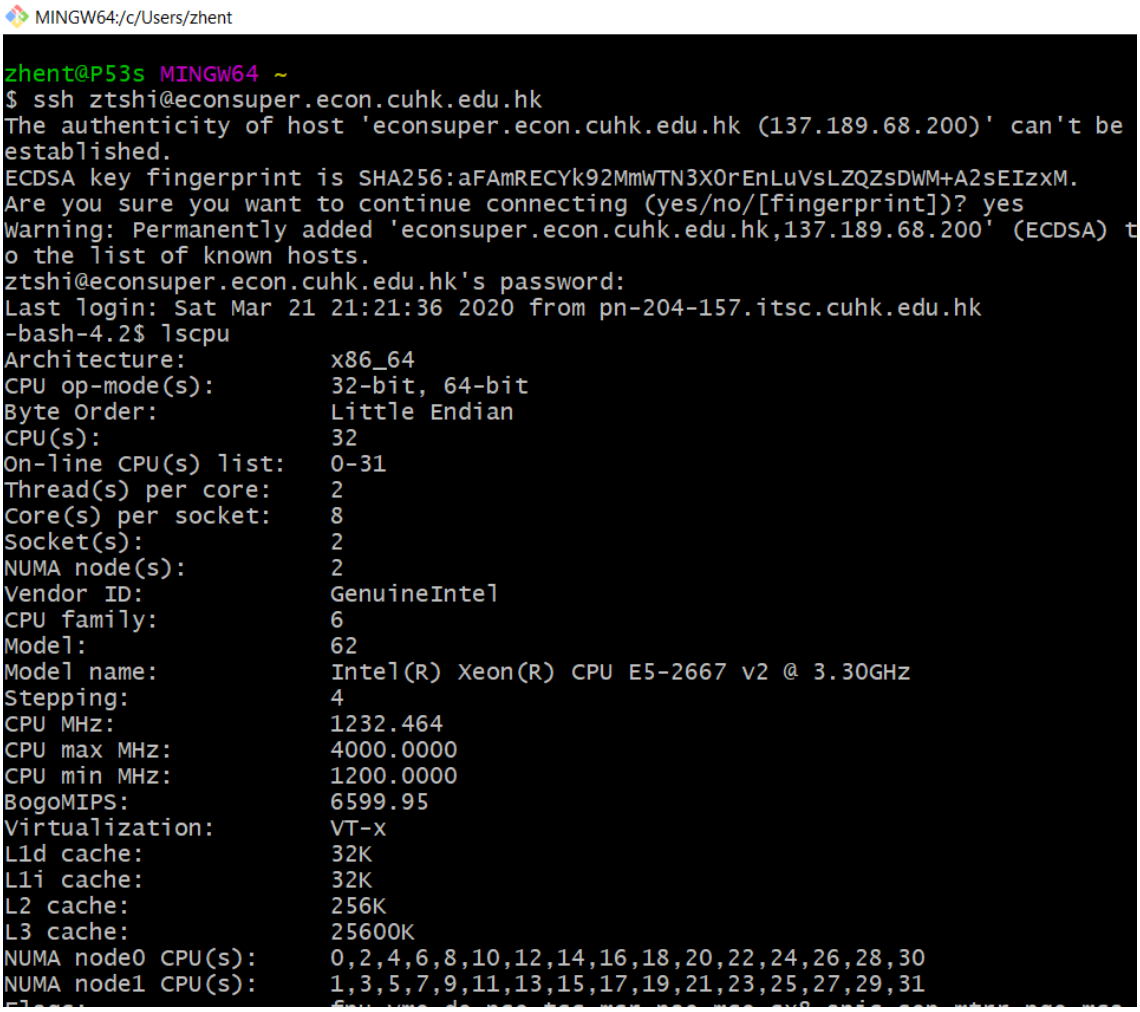


图 1: Log into `econsuper` and check CPU with `lscpu`

```
ssh ztshi@econsuper.econ.cuhk.edu.hk
cd data_example
R --vanilla <Beijing_housing_gbm.R> GBM_BJ.out &
```

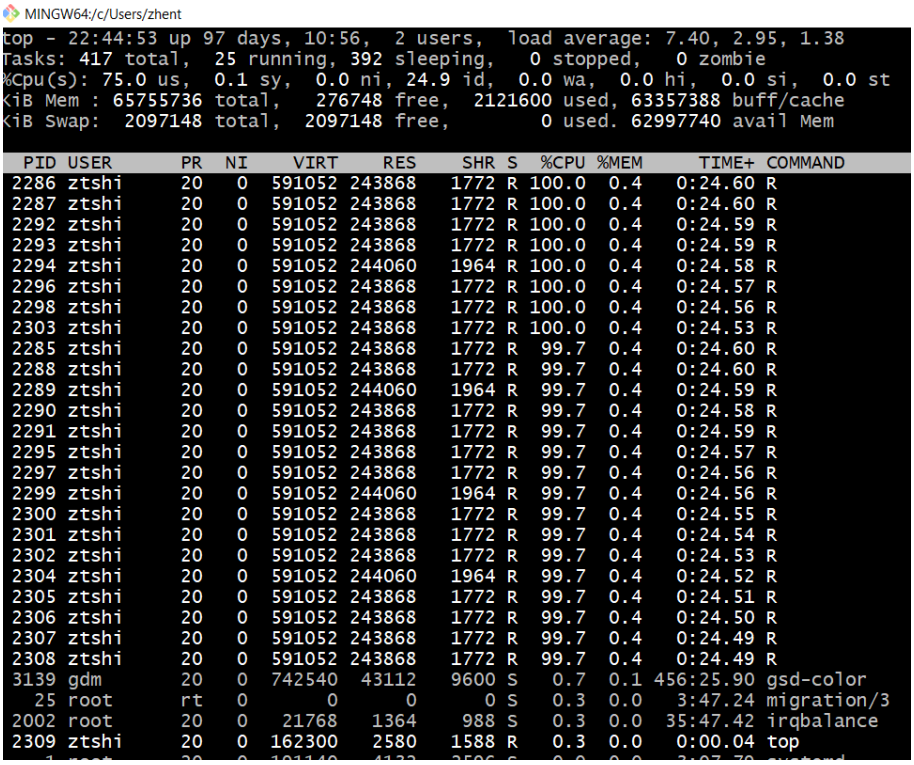


图 2: Running 24 cores on `econsuper`

2.5.2 RStudio Server

The command line shells lack a graphic interface for interactive data analysis. RStudio server offers a local-like environment via a web browser to communicate with a remote server. The remote server can be specified for users' need.

- `RStudio Cloud` is a free service to facilitate teaching and demonstration. The underlying computation unit is too weak to execute serious tasks.
- `Econsuper` is our department's service, which resembles a workplace environment in a small company. We can contact the technicians for our needs. The service is always online (with VPN connection), and much more powerful than the best local computer we can afford.
- `Amazon Web Service Cloud` is commercial service that can be tailored according to one's budget, from tiny demonstrative display to big enterprise business applications.

2.6 Graphics

An English clich茅 says "One picture is worth ten thousand words". John Tukey, a renowned mathematical statistician, was one of the pioneers of statistical graphs in the computer era. Nowadays,

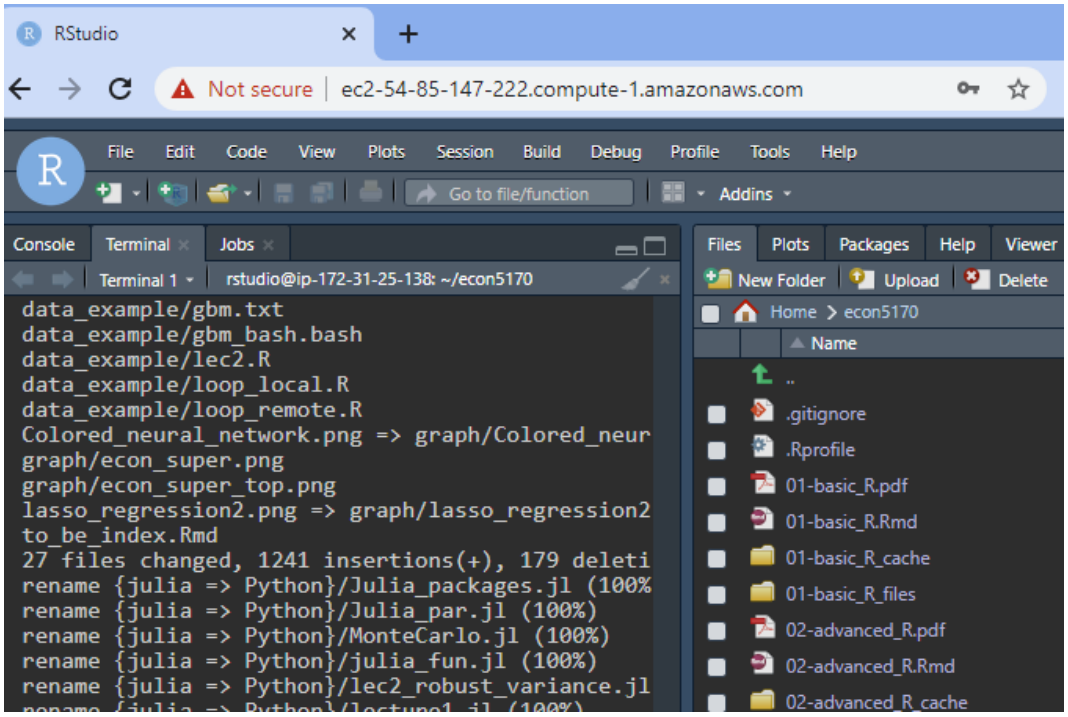


图 3: RStudio on AWS `t2.large` (2 CPUs, 8G Memory)

powerful software is able to produce dazzling statistical graphs, sometimes web-based and interactive. Outside of academia, journalism hooks a wide readership with professional data-based graphs. New York Times and The Economists are first-rate examples; South China Morning Post sometimes also does a respectable job. A well designed statistical graph can deliver an intuitive and powerful message. I consider graph prior to table when writing a research report or an academic paper. Graph is lively and engaging. Table is tedious and boring.

We have seen an example of R graph in the OLS regression linear example in Lecture 1. `plot` is a generic command for graphs, and is the default R graphic engine. It is capable of producing preliminary statistical graphs.

Over the years, developers all over the world have had many proposals for more sophisticated statistical graphs. Hadley Wickham's `ggplot2` is among the most successful.

`ggplot2` is an advanced graphic system that generates high-quality statistical graphs. It is not possible to cover it in a lecture. Fortunately, the author wrote a comprehensive reference **ggplot2 book**, which can be downloaded via the CUHK campus network (VPN needed).

`ggplot2` accommodates data frames of a particular format. `reshape2` is a package that helps prepare the data frames for `ggplot2`.

The workflow of ggplot is to add the elements in a graph one by one, and then print out the graph all together. In contrast, `plot` draws the main graph at first, and then adds the supplementary elements later.

`ggplot2` is particularly good at drawing multiple graphs, either of the same pattern or of different patterns. Multiple subgraphs convey rich information and easy comparison.

Example

Plot the density of two estimators under three different data generating processes. This is an example to generate subgraphs of the same pattern.

```
load("data_example/big150.Rdata")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.1.3
```

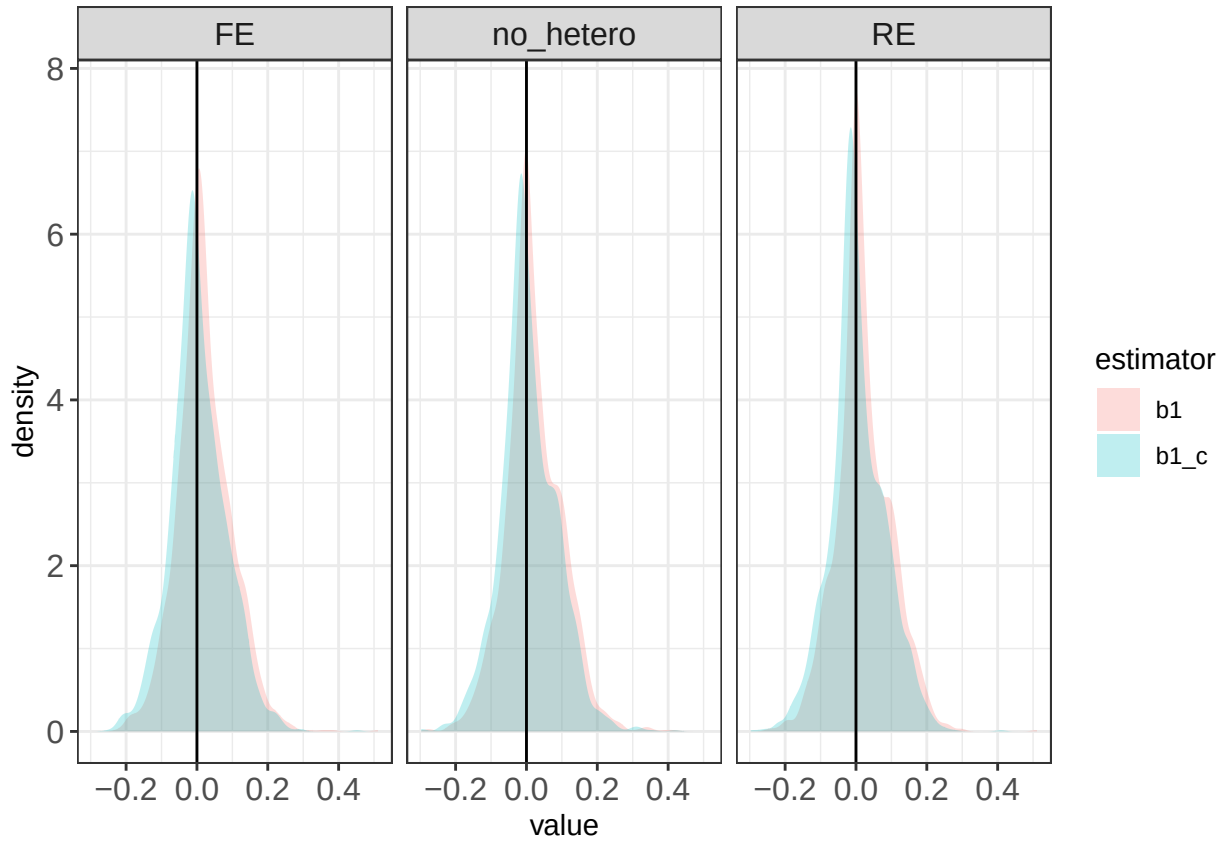
```
big150_1 <- big150[, c("typb", "b1", "b1_c")]
print(head(big150_1))
```

```
##       typb           b1        b1_c
## 12001   FE  0.124616242  0.11690387
## 12002   FE  0.267670157  0.25202802
## 12003   FE -0.030689329 -0.03976746
## 12004   FE  0.121169923  0.11866138
## 12005   FE  0.008300031 -0.02399673
## 12006   FE -0.026199118 -0.05231120
```

```
big150_1 <- melt(big150_1, id.vars = "typb", measure.vars = c("b1", "b1_c"))
names(big150_1)[2] <- c("estimator")
print(head(big150_1))
```

```
##   typb estimator        value
## 1   FE        b1  0.124616242
## 2   FE        b1  0.267670157
## 3   FE        b1 -0.030689329
## 4   FE        b1  0.121169923
## 5   FE        b1  0.008300031
## 6   FE        b1 -0.026199118
```

```
p1 <- ggplot(big150_1)
p1 <- p1 + geom_area(
  stat = "density", alpha = .25,
  aes(x = value, fill = estimator), position = "identity"
)
p1 <- p1 + facet_grid(. ~ typb)
p1 <- p1 + geom_vline(xintercept = 0)
p1 <- p1 + theme_bw()
p1 <- p1 + theme(
  strip.text = element_text(size = 12),
  axis.text = element_text(size = 12)
)
print(p1)
```



The function `ggplot` specifies which dataset to use for the graph. `geom_***` determines the shape to draw, for example scatter dots, lines, curves or areas. `theme` is to tune the supplementary elements like the background, the size and font of the axis text and so on.

Example

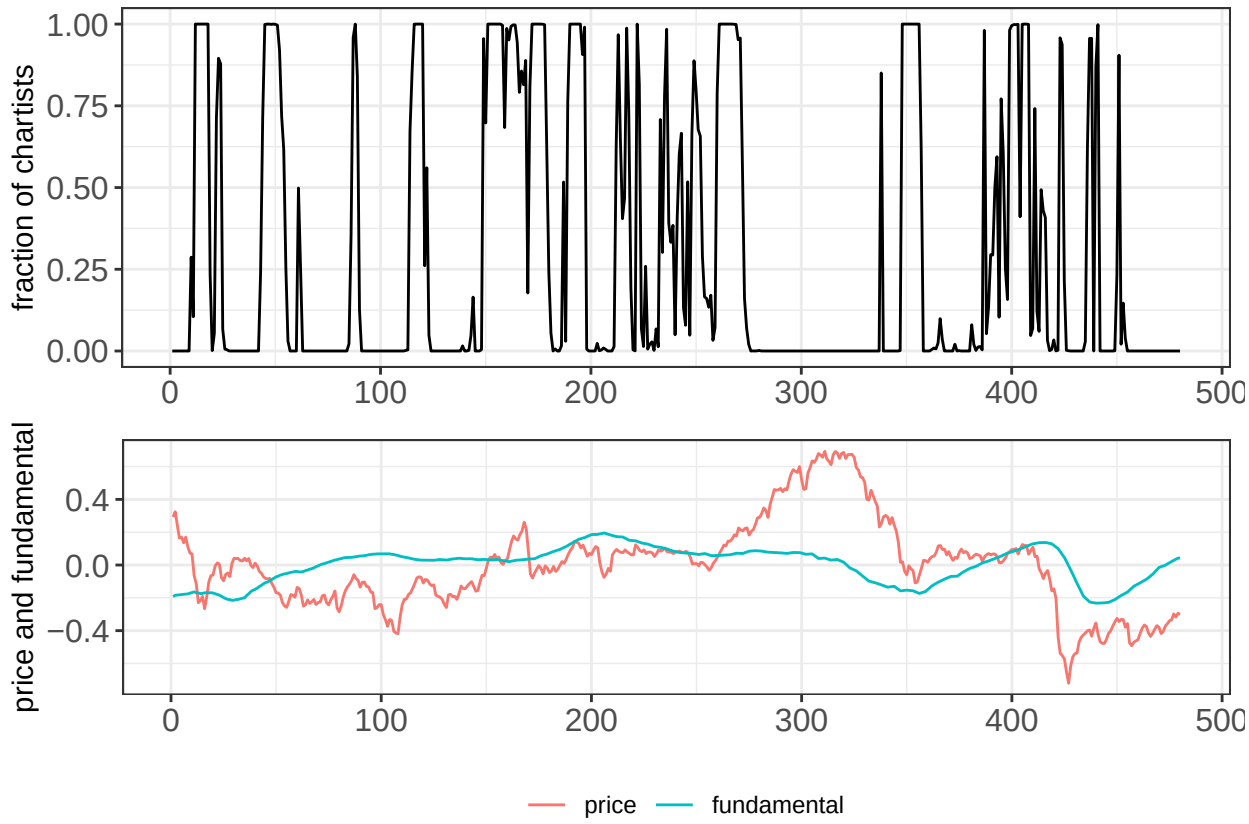
This example aligns two graphs of different patterns in one page. Similar graphs appear in Shi and Zheng (2018).

```
# graph packages
library(lattice)
library(ggplot2)
library(reshape2)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.1.3
```

```
load("data_example/multigraph.Rdata") # load data


# unify the theme in the two graphs
theme1 <- theme_bw() + theme(
  axis.title.x = element_blank(),
  strip.text = element_text(size = 12),
  axis.text = element_text(size = 12),
```

```
  legend.position = "bottom", legend.title = element_blank()
)

# sub-graph 1
d1 <- data.frame(month = 1:480, m = m_vec)
p1 <- qplot(x = month, y = m, data = d1, geom = "line")
p1 <- p1 + theme1 + ylab("fraction of chartists")


# sug-graph 2
d2$month <- 1:480
p2 <- ggplot(d2)
p2 <- p2 + geom_line(aes(x = month, y = value, col = variable))
p2 <- p2 + theme1 + ylab("price and fundamental")

# generate the grahp
grid.arrange(p1, p2, nrow = 2)
```



In order to unify the theme of the two distinctive subgraphs, we define an object `theme1` and apply it in both graphic objects `p1` and `p2`.

2.6.1 Interactive Graph

In the folder of `data_example`, we give a preliminary example of `flexdashboard`. It is very easy to convert a ggplot2 graph into an HTML interactive graph by `plotly::ggplotly()`.

2.6.2 Future writing plan

- Shiny app tutorial. I have included a shiny app in `data_example/shiny`.

2.7 Reading

Wickham and Grolemund: Ch 3, 10, 11, 21, and 26-30

2.8 References

3 Integration

In their mathematical definitions, integration and differentiation involve taking limit. However, our computer is a finite-precision machine that can handle neither arbitrarily small nor arbitrarily large numbers; it can, at best, approximate the limiting behavior. In this lecture, we first briefly talk about numerical differentiation and integration, and then we discuss stochastic methods with examples from econometrics. In particular, we will introduce simulated method of moments, indirect inference, and Markov Chain Monte Carlo (MCMC). These methods are beyond the in-class coverage of Econ5121A and Econ5150. Interested readers are referred to Cameron and Trivedi (2005) (Chapters 12 and 13) for details.

3.1 Numerical Methods

Numerical differentiation and integration are fundamental topics and of great practical importance. However, how the computer works out these operations has nothing to do with economics or econometrics; it is the content of a numerical analysis course. Here we quickly go over the numerical methods. Judd (1998) (Chapter 7) is an authoritative reference.

In undergraduate calculus, we have learned the analytic differentiation of many common functions. However, there are cases in which analytic forms are unavailable or too cumbersome to program. For instance, to find the optimum for the objective function $f : R^K \mapsto R$ by Newton's method, in principle we need to code the K-dimensional gradient and the $K \times K$-dimensional Hessian matrix. Programming up the gradient and the Hessian manually is a time-consuming and error-prone job. What is worse, whenever we change the objective function, which happens often at the experimental stage of research, we have to redo the gradient and Hessian. Therefore, it is more efficient to use numerical differentiation instead of coding up the analytical expressions, in particular in the trial-and-error stage.

The partial derivative of a multivariate function $f : R^K \mapsto R$ at a point $x_0 \in R^K$ is

$$\frac{\partial f(x)}{\partial x_k}\Big|_{x=x_0} = \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon \cdot e_k) - f(x_0 - \epsilon \cdot e_k)}{2\epsilon},$$

where $e_k = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the identifier of the k-th coordinate. The numerical execution in a computer follows the basic definition to evaluate $f(x_0 \pm \epsilon \cdot e_k))$ with a small ϵ. But how small is small? Usually we try a sequence of ϵ's until the numerical derivative is stable. There are also more sophisticated algorithms.

In R, the package `numDeriv` conducts numerical differentiation, in which

- `grad` for a scalar-valued function;
- `jacobian` for a real-vector-valued function;
- `hessian` for a scalar-valued function;
- `genD` for a real-vector-valued function.

Integration is, in general, more difficult than differentiation. In R, `integrate` carries out one-dimensional quadrature, and `adaptIntegrate` in the package `cubature` deals with multi-dimensional quadrature. The reader is referred to the documentation for the algorithm behind numerical integrations.

Numerical methods are not panacea. Not all functions are differentiable or integrable. Before turning to numerical methods, it is always imperative to try to understand the behavior of the function at the first place. Some symbolic software, such as `Mathematica` or `Wolfram Alpha`, is a useful tool for this purpose. R is weak in symbolic calculation despite the existence of a few packages for this purpose.

3.2 Stochastic Methods

An alternative to numerical integration is the stochastic methods. The underlying principle of stochastic integration is the law of large numbers. Let $\int h(x)dF(x)$ be an integral where $F(x)$ is a probability distribution. We can approximate the integral by $\int h(x)dF(x) \approx S^{-1} \sum_{s=1}^{S} h(x_s)$, where x_s is randomly generated from $F(x)$. When S is large, a law of large numbers gives

$$S^{-1} \sum_{s=1}^{S} h(x_s) \xrightarrow{\text{P}} E[h(x)] = \int h(x)dF(x).$$

If the integration is carried out not in the entire support of $F(x)$ but on a subset A, then

$$\int_A h(x)dF(x) \approx S^{-1} \sum_{s=1}^{S} h(x_s) \cdot 1\{x_s \in A\},$$

where $1\{\cdot\}$ is the indicator function.

In theory, we want to use an S as large as possible. In reality, we are constrained by the computer's memory and computing time. There is no clear guidance of the size of S in practice. Preliminary experiment can help decide an S that produces stable results.

Stochastic integration is popular in econometrics and statistics, thanks to its convenience in execution.

Example

Structural econometric estimation starts from economic principles. In an economic model, some elements unobservable to the econometrician dictate an economic agent's decision. Roy (1951) proposes such a structural model with latent variables, and the Roy model is the foundation of self-selection in labor economics.

In the original paper of the Roy model, an economic agent must be either a farmer or a fisher. The utility of being a farmer is $U_1^\star = x'\beta_1 + e_1$ and that of being a fisher is $U_2^\star = x'\beta_2 + e_2$, where $U_1^\star$ and $U_2^\star$ are latent (unobservable). The econometrician observes the binary outcome $y = \mathbf{1}\{U_1^\star > U_2^\star\}$. If (e_1, e_2) is independent of x, and

$$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & 1 \end{bmatrix} \right)$$

where σ_2 is normalized to be 1, we can write down the log-likelihood as

$$L(\theta) = \sum_{i=1}^{n} \left\{ y_i \log P(U_{i1}^\star > U_{i2}^\star) + (1 - y_i) \log P(U_{i1}^* \leq U_{i2}^\star) \right\}.$$

Let $\theta = (\beta_1, \beta_2, \sigma_1, \sigma_{12})$. Given a trial value θ, we can compute

$$p(\theta | x_i) = P\left(U_{i1}^\star(\theta) > U_{i2}^\star(\theta)\right) = P\left(x_i'(\beta_1 - \beta_2) > e_{i2} - e_{i1}\right).$$

Under the joint normal assumption, $e_{i2} - e_{i1} \sim N(0, \sigma_1^2 - 2\sigma_{12} + 1)$ so that

$$p(\theta | x_i) = \Phi\left(\frac{x_i'(\beta_1 - \beta_2)}{\sqrt{\sigma_1^2 - 2\sigma_{12} + 1}} \right)$$

where $\Phi(\cdot)$ is the CDF of the standard normal.

However, notice that the analytical form depends on the joint normal assumption and cannot be easily extended to other distributions. As long as the joint distribution of (e_{i1}, e_{i2}), no matter it is normal or not, can be generated from the computer, we can use the stochastic method. We estimate

$$\hat{p}(\theta | x_i) = \frac{1}{S} \sum_{i=1}^{S} \mathbf{1}\left(U_{i1}^{s*}(\theta) > U_{i2}^{s*}(\theta)\right),$$

where $s = 1, \ldots, S$ is the index of simulation and S is the total number of simulation replications.

Next, we match moments generated the theoretical model with their empirical counterparts. The choice of the moments to be matched is to be decided by the user. A set of valid choice for the Roy model example is

$$g_1(\theta) = n^{-1}\sum_{i=1}^{n} x_i(y_i - \hat{p}(\theta|x_i)) \approx 0 \tag{1}$$

$$g_2(\theta) = n^{-1}\sum_{i=1}^{n} (y_i - \bar{y})^2 - \bar{\hat{p}}(\theta|x_i)(1 - \bar{\hat{p}}(\theta|x_i)) \approx 0 \tag{2}$$

$$g_3(\theta) = n^{-1}\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \hat{p}(\theta|x_i))^2 \approx 0 \tag{3}$$

where $\bar{y} = n^{-1}\sum_{i=1}^{n} y_i$ and $\bar{\hat{p}}(\theta) = n^{-1}\sum_{i=1}^{n} p(\theta|x_i)$. The first set of moments is justified by the independence of (e_{i1}, e_{i2}) and x_i so that $E[x_i y_i] = x_i E[y_i|x_i] = x_i p(\theta|x_i)$, and the second set matches the variance of y_i. Since the moment conditions $(g_j(\theta))_{j=1}^{3}$ equals the number of unknown parameters, these moment conditions just-identifies the parameter θ. We need to come up with more moment conditions for over-identification. Moreover, we need to choose a weighting matrix W to form a quadratic criterion for GMM in over-identification.

The above example can be viewed as an application of simulated maximum likelihood. In parallel, we can simulate a moment condition if its explicit form is unavailable. Pakes and Pollard (1989) provide the theoretical foundation of the simulated method of moments (SMM). Our co-teacher Prof. Guo (Guo, Xia, and Zhang 2018) recently applies SMM to estimate a structural labor model.

3.2.1 Indirect Inference

Indirect inference Gourieroux, Monfort, and Renault (1993) is yet another simulated-based estimation method. Indirect inference is extensively used in structural model estimation (T. Li 2010). Theoretical analysis of indirect inference reveals its nice properties in bias deduction via a proper choice of the binding function (Phillips 2012).

The basic idea of indirect inference is to recover the structural parameter from an *auxiliary model*—usually an reduced-form regression. The reduced-form regression ignores the underlying economic structure and is a purely statistical procedure; thus the reduced-form regression is relatively easier to implement. A *binding function* is a one-to-one mapping from the parameter space of the reduced-form to that of the structural form. Once the reduced-form parameter is estimated, we can recover the structural parameter via the binding function. When the reduced-form parameter can be expressed in closed-form, we can utilize the analytical form to match the theoretical prediction and the empirical outcomes, as in Shi and Zheng (2018). In most cases however, the reduced-form implied by the structural model does not have a closed-form expression so simulation becomes necessary.

The choice of the auxiliary model is not unique. In the Roy model example where θ is the structural parameter, a sensible starting point to construct the auxiliary model is the linear regression between y_i and x_i. A set of reduced-form parameters can be chosen as $\hat{b} = (\hat{b}_1, \hat{b}_2, \hat{b}_3)'$, where

$$\hat{b}_1 = (X'X)^{-1}X'y \tag{4}$$
$$\hat{b}_2 = n^{-1}\sum_{y_i=1}(y_i - x_i'b_1)^2 = n^{-1}\sum_{y_i=1}(1 - x_i'b_1)^2 \tag{5}$$
$$\hat{b}_3 = n^{-1}\sum_{y_i=0}(y_i - x_i'b_1)^2 = n^{-1}\sum_{y_i=0}(x_i'b_1)^2. \tag{6}$$

Here $\hat{b}_1$ is associated with β, and $(\hat{b}_2, \hat{b}_3)$ are associated with (σ_1, σ_{12}).

Now we consider the structural parameter. Given a trial value θ, the model is parametric and we can simulate artificial error (e_{i1}^*, e_{i2}^*) conditional on x_i. In each simulation experiment, we can decide y_i^*, and we can further estimate the reduced-form parameter $\hat{b}^* = \hat{b}^*(\theta)$ given the artificial data. $b(\theta)$ is the binding function. Conducting such simulation for S times, we measure the distance between $\hat{b}$ and $\hat{b}^*$ as

$$Q(\theta) = \left(\hat{b} - S^{-1}\sum_{s=1}^{S}\hat{b}^*(\theta)^s\right)' W \left(\hat{b} - S^{-1}\sum_{s=1}^{S}\hat{b}^*(\theta)^s\right)$$

where s indexes the simulation and W is a positive definite weighting matrix. The indirect inference estimator is $\hat{\theta} = \arg\min_\theta Q(\theta)$. That is, we seek the value of θ that minimizes the distance between the reduced-form parameter from the real data and that from the simulated artificial data.

3.3 Markov Chain Monte Carlo

If the CDF $F(X)$ is known, it is easy to generate random variables that follow such a distribution. We can simply compute $X = F^{-1}(U)$, where U is a random draw from Uniform$(0, 1)$. This X follows the distribution $F(X)$.

If the pdf $f(X)$ is known, we can generate a sample with such a distribution by *importance sampling*. Metropolis-Hastings algorithm (MH algorithm) is such a method. MH is one of the Markov Chain Monte Carlo methods. It can be implemented in the R package `mcmc`. This page contains demonstrative examples of MCMC.

3.3.1 Metropolis-Hastings Algorithm

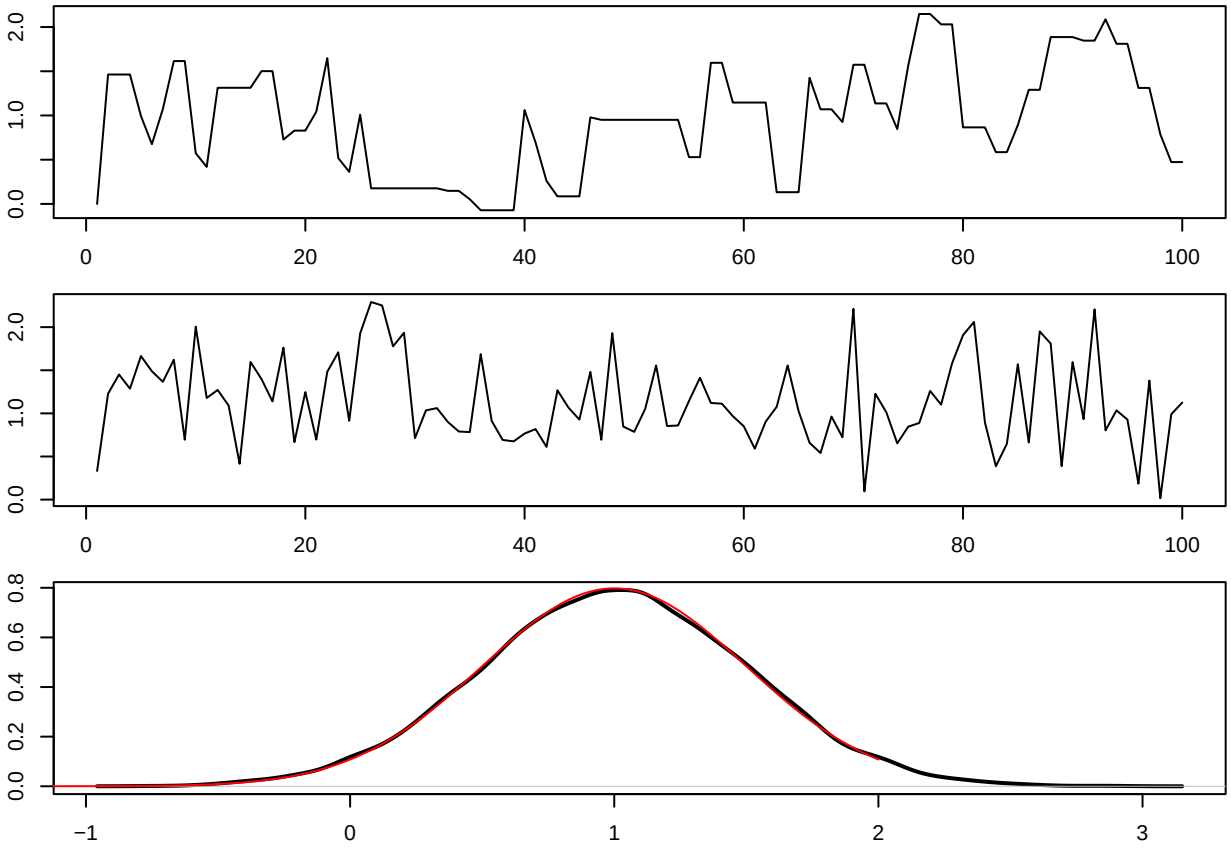
The underlying theory of the MH requires long derivation, but implementation is straightforward. Here we use MH to generate a sample of normally distributed observations with $\mu = 1$ and $\sigma = 0.5$. In the function `metrop`, we provide the logarithm of the density of

$$\log f(x) = -\frac{1}{2}\log(2\pi) - \log\sigma - \frac{1}{2\sigma^2}(x - \mu)^2,$$

and the first term can be omitted as it is irrelevant to the parameter.

```
h <- function(x, mu = 1, sd = 0.5) {
  y <- -log(sd) - (x - mu)^2 / (2 * sd^2)
} # un-normalized function (doesn't need to integrate as 1)
```

```r
out <- mcmc::metrop(obj = h, initial = 0, nbatch = 100, nspac = 1)

par(mfrow = c(3, 1))
par(mar = c(2, 2, 1, 1))
plot(out$batch, type = "l") # a time series with flat steps

out <- mcmc::metrop(obj = h, initial = 0, nbatch = 100, nspac = 10)
plot(out$batch, type = "l") # a time series looks like a white noise

out <- mcmc::metrop(obj = h, initial = 0, nbatch = 10000, nspac = 10)
plot(density(out$batch), main = "", lwd = 2)

xbase <- seq(-2, 2, 0.01)
ynorm <- dnorm(xbase, mean = 1, sd = 0.5)
lines(x = xbase, y = ynorm, type = "l", col = "red")
```



The generated graph consists of three panels. The first panel is a time series where the marginal distribution of each observations follows $N(1, 0.5^2)$. The time dependence is visible, and flat regions are observed when the Markov chain rejects a new proposal so the value does not update over two periods. To reduced time dependence, the middle panel collects the time series every 10 observations on the Markov chain. No flat region is observed in this subgraph and the serial correlation is weakened. The third panel compares the kernel density of the simulated observations (black curve)

with the density function of $N(1, 0.5^2)$ (red curve).

3.3.2 Laplace-type Estimator: An Application of MCMC

For some econometric estimators, finding the global optimizer is known to be difficult, because of irregular behavior of the objective function. Chernozhukov and Hong (2003)'s *Laplace-type estimator* (LTE), or Quasi-Bayesian estimator (QBE), is an alternative to circumvent the challenge in optimization. LTE transforms the value of the criterion function of an extremum estimator into a probability weight

$$f_n(\theta) = \frac{\exp(-L_n(\theta))\pi(\theta)}{\int_\Theta \exp(-L_n(\theta))\pi(\theta)}$$

where $L_n(\theta)$ is an criterion function (say, OLS criterion, (negative) log likelihood criterion, or GMM criterion), and $\pi(\theta)$ is the density of a prior distribution. The smaller is the value of the objective function, the larger it weighs. The exponential transformation comes from Laplace approximation.

We use MCMC to simulate the distribution of θ. From a Bayesian's viewpoint, $f_n(\theta)$ is the posterior distribution. However, Chernozhukov and Hong (2003) use this distribution for classical estimation and inference, and they justify the procedure via frequentist asymptotic theory. Once $f_n(\theta)$ is known, then *asymptotically* the point estimator equals its mean under the quadratic loss function, and equals its median under the absolute-value loss function.

The code block below compares the OLS estimator with the LTE estimator in a linear regression model.

```
library(magrittr)
```

```
## Warning: package 'magrittr' was built under R version 4.1.3
```

```
# DGP
n <- 500
b0 <- c(.1, .1)
X <- cbind(1, rnorm(n))
Y <- X %*% b0 + rnorm(n)

b2_OLS <- (lm(Y ~ -1 + X) %>% summary() %>% coef())[2, ]
# "-1" in lm( ) because X has contained intercept
print(cat(
  "The OLS point est =", b2_OLS[1], " sd = ", b2_OLS[2],
  " \n and C.I.=(", c(b2_OLS[1] - 1.96 * b2_OLS[2], b2_OLS[1] + 1.96 * b2_OLS[2]), ")"
))
```

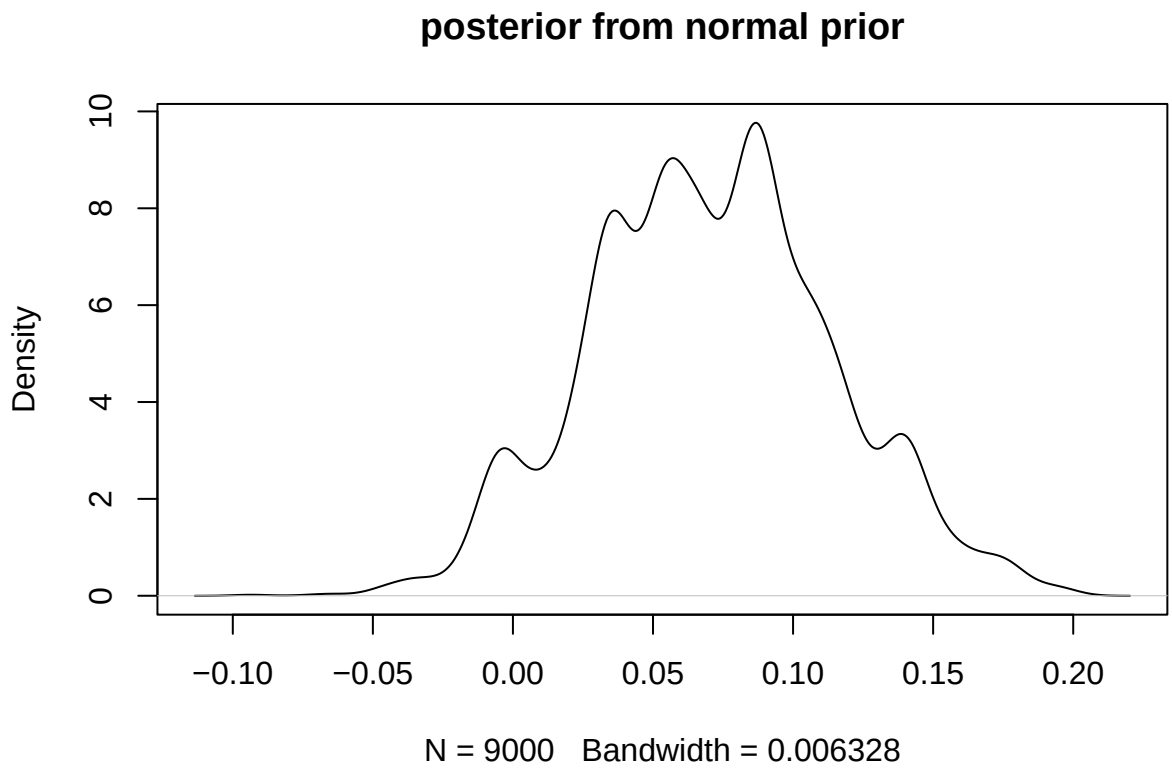
```
## The OLS point est = 0.06994462  sd =  0.04269889
##  and C.I.=( -0.0137452 0.1536344 )NULL
```

```
# Laplace-type estimator
L <- function(b) -0.5 * sum((Y - X %*% b)^2) - 0.5 * crossprod(b - c(0, 0))
# notice the "minus" sign of the OLS objective function
# here we use a normal prior around (0,0).
# results are very similar if we replace it with a flat prior so that
```

```r
# L <- function(b) -0.5*sum((Y - X %*% b)^2)

nbatch <- 10000
out <- mcmc::metrop(obj = L, initial = c(0, 0), nbatch = nbatch, nspac = 20)

# summarize the estimation
bhat2 <- out$batch[-(1:round(nbatch / 10)), 2] # remove the burn in
bhat2_point <- mean(bhat2)
bhat2_sd <- sd(bhat2)
bhat2_CI <- quantile(bhat2, c(.025, .975))

# compare with OLS
print(cat(
  "The posterior mean =", bhat2_point, " sd = ", bhat2_sd,
  " \n and C.I.=(", bhat2_CI, ")"
))
```

```
## The posterior mean = 0.07102107  sd =  0.04343509
##  and C.I.=( -0.0101997 0.1599032 )NULL
```

```r
plot(density(bhat2), main = "posterior from normal prior")
```



3.4 Future Writing

- EM algorithm

3.5 Reading

- T. Li (2010)
- Peng (2018), Advanced Statistical Computing, Ch. 5, Ch. 7.1-2

3.6 References

4 Simulation

```
library(magrittr)
```

```
## Warning: package 'magrittr' was built under R version 4.1.3
```

```
library(tibble)
```

```
## Warning: package 'tibble' was built under R version 4.1.3
```

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 4.1.3
```

```
set.seed(888)
```

Probability theory has an infamous inception for its association with gambling. Monte Carlo, the European casino capital, is another unfortunate presence. However, naming it Macau simulation or Hong Kong Jockey Club simulation does not make me feel any better. I decide to simply call it "simulation".

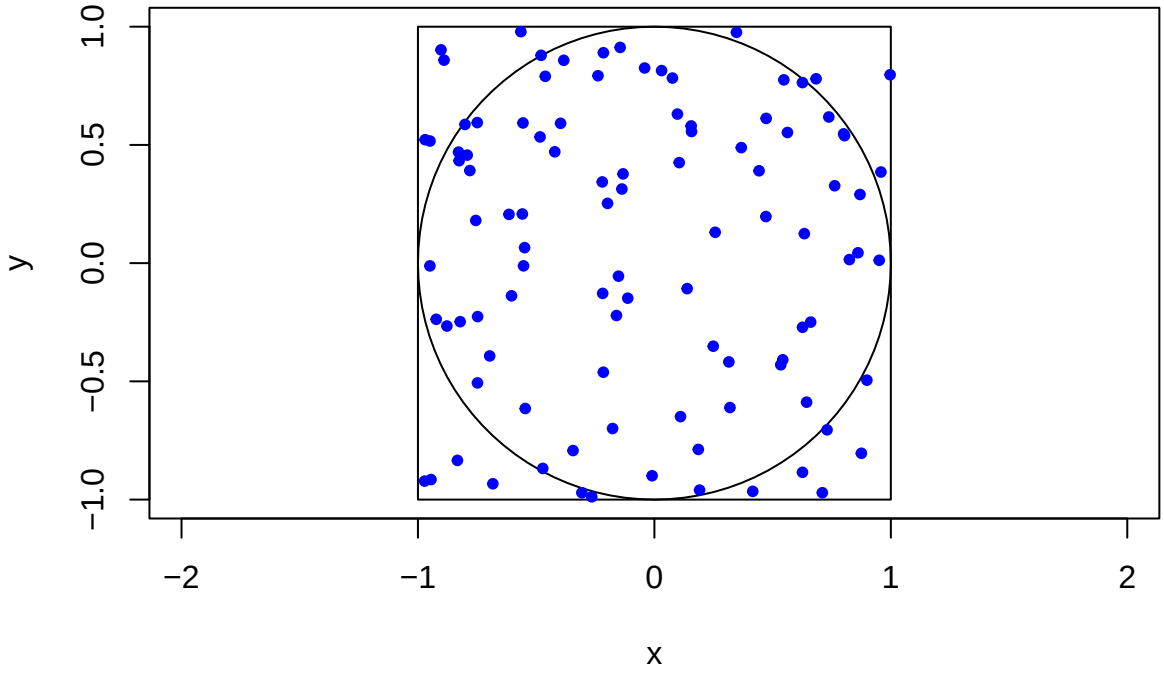
Simulation has been widely used for (i) checking finite-sample performance of asymptotic theory; (ii) bootstrap, an automated data-driven inference procedure; (iii) generating non-standard distributions; (iv) approximating integrals with no analytic expressions. In this lecture, we will focus on (i) and (ii), whereas (iii) and (iv) will be deferred to the next lecture on integration.

From now on, we will start to write script. A script is a piece of code for a particular purpose. A script of thousands of lines is not written from the beginning to the end; we develop it recursively. We cut a big job into small manageable tasks. Write a small piece, test it, and encapsulate it into a user-defined function whenever necessary. Small pieces are integrated by the super structure. This is just like building an Airbus 380. The engines and wings are made in UK, the fuselage is made in Germany and so on. All pieces are assembled in Toulouse, France, and then the giant steel bird can fly. Finally, add comments to the script to facilitate readability. Without comments you will forget what you did when you open the script again one month later.

Example

Zu Chongzhi (429–500 AD), an ancient Chinese mathematician, calculated π being between 3.1415926 and 3.1415927, which for 900 years held the world record of the most accurate π. He used a deterministic approximation algorithm. Now imagine that we present to Zu Chongzhi, with full respect and admiration, a modern computer. How can he achieve a better approximation?

```
require(plotrix)
require(grid)

plot(c(-1, 1), c(-1, 1), type = "n", asp = 1, xlab = "x", ylab = "y")
rect(-1, -1, 1, 1)
draw.circle(0, 0, 1)
points(x = runif(100) * 2 - 1, y = runif(100) * 2 - 1, pch = 20, col = "blue")
```



Standing on the shoulder of laws of large numbers, π can be approximated by a stochastic algorithm. Since

$$\frac{\text{area of a circle}}{\text{area of the smallest encompassing square}} = \frac{\pi r^2}{(2r)^2} = E\left[\mathbf{1}\left\{x^2 + y^2 \leq 1\right\}\right],$$

it implies $\pi = 4 \times E[1\{x^2 + y^2 \leq 1\}]$. The mathematical expectation is unknown, and it can be approximated by simulation.

```
n <- 10000000
Z <- 2 * matrix(runif(n), ncol = 2) - 1 # uniform distribution in [-1, 1]

inside <- mean(sqrt(rowSums(Z^2)) <= 1) # the center of the cirle is (0, 0)
```

```
cat("The estimated pi = ", inside * 4, "\n")
```

```
## The estimated pi =  3.142574
```

The sample size can be made as large as the computer's memory permits, and we can iterate it with average of averages and so on, for higher accuracy.

4.1 Finite Sample Evaluation

In the real world, a sample is finite. The distribution of a statistic in finite sample depends on the sample size n, which has only in rare cases a simple mathematical expression. Fortunately, the expression can often be simplified when we imagine the sample size being arbitrarily large. Asymptotic theory is such mathematical apparatus to approximate finite sample distributions. It is so far the most useful analytical tool that helps us understand the behavior of estimators and tests, either in econometrics or in statistics in general. Simulation is one way to evaluate the accuracy of approximation.

Even though real-data empirical example can also be used to illustrate a statistical procedure, artificial data are convenient and informative. The prevalent paradigm in econometrics is to assume that the data are generated from a model. We, as researchers, check how close the estimate is to the model characterized by a set of unknown parameters. In simulations we have full control of the data generation process, including the true parameter. In a real example, however, we have no knowledge about the true model, so we cannot directly evaluate the quality of parameter estimation.

(It would be a different story if we are mostly interested in prediction, as we often encounter in machine learning. In such cases, we can split the data into two parts: one part for modeling and estimation, and the other for verification.)

Example

In OLS theory, the classical views X as fixed regressions and only cares about the randomness of the error term. Modern econometrics textbook emphasizes that a random X is more appropriate for econometrics applications. In rigorous textbooks, the moment of X is explicitly stated as $E[X_i X_i'] < \infty$. *Is asymptotic inferential theory for the OLS estimator—consistency and asymptotic normality—valid when X follows a Pareto distribution with shape coefficient 1.5?* A Pareto distribution with shape coefficient between 1 and 2 has finite population mean but infinite variance. Therefore this case violates the assumptions for OLS stated in most of econometric textbooks.

We write a script to investigate this problem. The following steps develop the code.

1. given a sample size, get the OLS `b_hat` and its associated `t_value`.
2. wrap `t_value` as a user-defined function so that we can reuse it for many times.
3. given a sample size, report the size under two distributions.
4. wrap step 3 again as a user-defined function, ready for different sample sizes.
5. develop the super structure to connect the workhorse functions.
6. add comments and documentation.

```
# the workhorse functions
simulation <- function(n, type = "Normal", df = df) {
```

```r
  # a function gives the t-value under the null
  if (type == "Normal") {
    e <- rnorm(n)
  } else if (type == "T") {
    e <- rt(n, df)
  }

  X <- cbind(1, VGAM::rpareto(n, shape = 1.5))
  Y <- X %*% b0 + e
  rm(e)

  bhat <- solve(t(X) %*% X, t(X) %*% Y)
  bhat2 <- bhat[2] # parameter we want to test

  e_hat <- Y - X %*% bhat
  sigma_hat_square <- sum(e_hat^2) / (n - 2)
  sig_B <- solve(t(X) %*% X) * sigma_hat_square
  t_value_2 <- (bhat2 - b0[2]) / sqrt(sig_B[2, 2])

  return(t_value_2)
}

# report the empirical test size
report <- function(n) {
  # collect the test size from the two distributions
  # this function contains some repetitive code, but is OK for such a simple one
  TEST_SIZE <- rep(0, 3)

  # e ~ normal distribution, under which the t-dist is exact
  Res <- plyr::ldply(.data = 1:Rep, .fun = function(i) simulation(n, "Normal"))
  TEST_SIZE[1] <- mean(abs(Res) > qt(.975, n - 2))
  TEST_SIZE[2] <- mean(abs(Res) > qnorm(.975))

  # e ~ t-distribution, under which the exact distribution is complicated.
  # we rely on asymptotic normal distribution for inference instead
  Res <- plyr::ldply(.data = 1:Rep, .fun = function(i) simulation(n, "T", df))
  TEST_SIZE[3] <- mean(abs(Res) > qnorm(.975))

  return(TEST_SIZE)
}

## the super structure
# set the parameters
Rep <- 1000
b0 <- matrix(1, nrow = 2)
df <- 1 # t dist. with df = 1 is Cauchy
```

```
# run the calculation of the empirical sizes for different sample sizes
NN <- c(5, 10, 200, 5000)
RES <- plyr::ldply(.data = NN, .fun = report)
names(RES) <- c("exact", "normal.asym", "cauchy.asym") # to make the results readable
RES$n <- NN
RES <- RES[, c(4, 1:3)] # beautify the print
print(RES)
```

```
##      n exact normal.asym cauchy.asym
## 1    5 0.048       0.146       0.165
## 2   10 0.036       0.079       0.096
## 3  200 0.043       0.044       0.034
## 4 5000 0.046       0.046       0.027
```

The first column is the when the error is normal, and we use the exactly distribution theory to find the critical value (according to the t distribution.) The second column still uses the normal distribution in the error term, but change the critical value to be from the normal distribution, which is based on asymptotic approximation. When sample size is small, obvious size distortion is observed; but the deviation is mitigated when the sample size increases. When the error distribution is Cauchy, the so called "exact distribution" is no longer exact— it is exact only if the error is normal and independent from x. If we attempt to use the asymptotic normal approximation, we find that the asymptotic approximation breaks down. The test size does not converge to the nominal 5% as the sample size increases.

In this simulation design, X is always Pareto while we vary the distribution of the error term. When we look at the table, we witness that the distribution of X indeed does not matter. Why? Since

$$\sqrt{n}(\hat{\beta} - \beta_0)|X = (X'X/n)^{-1}(X'e/\sqrt{n}),$$

the k-th element of the vector coefficient conditional on X is

$$\widehat{\beta}_k|X = \eta_k'\widehat{\beta}|X \sim N\left(\beta_k, \sigma^2 \left(X'X\right)_{kk}^{-1}\right).$$

The t-statistic

$$T_k = \frac{\widehat{\beta}_k - \beta_k}{\sqrt{s^2\left[(X'X)^{-1}\right]_{kk}}} = \frac{\widehat{\beta}_k - \beta_k}{\sqrt{\sigma^2\left[(X'X)^{-1}\right]_{kk}}} \cdot \frac{\sqrt{\sigma^2}}{\sqrt{s^2}} = \frac{\left(\widehat{\beta}_k - \beta_k\right)/\sqrt{\sigma^2\left[(X'X)^{-1}\right]_{kk}}}{\sqrt{\frac{e'}{\sigma}M_X\frac{e}{\sigma}/\left(n-K\right)}}.$$

Even though $X'X/n$ does not converge to a stable probabilistic limit, the self-normalized t statistic does not break down. Regardless the distribution of X, when the error term is normal, the numerator of the above expression follows a standard normal distribution and the demonimator follows a χ^2, and moreover these two components are independent. The resulting statistic follows a t-distribution.

4.2 Bootstrap

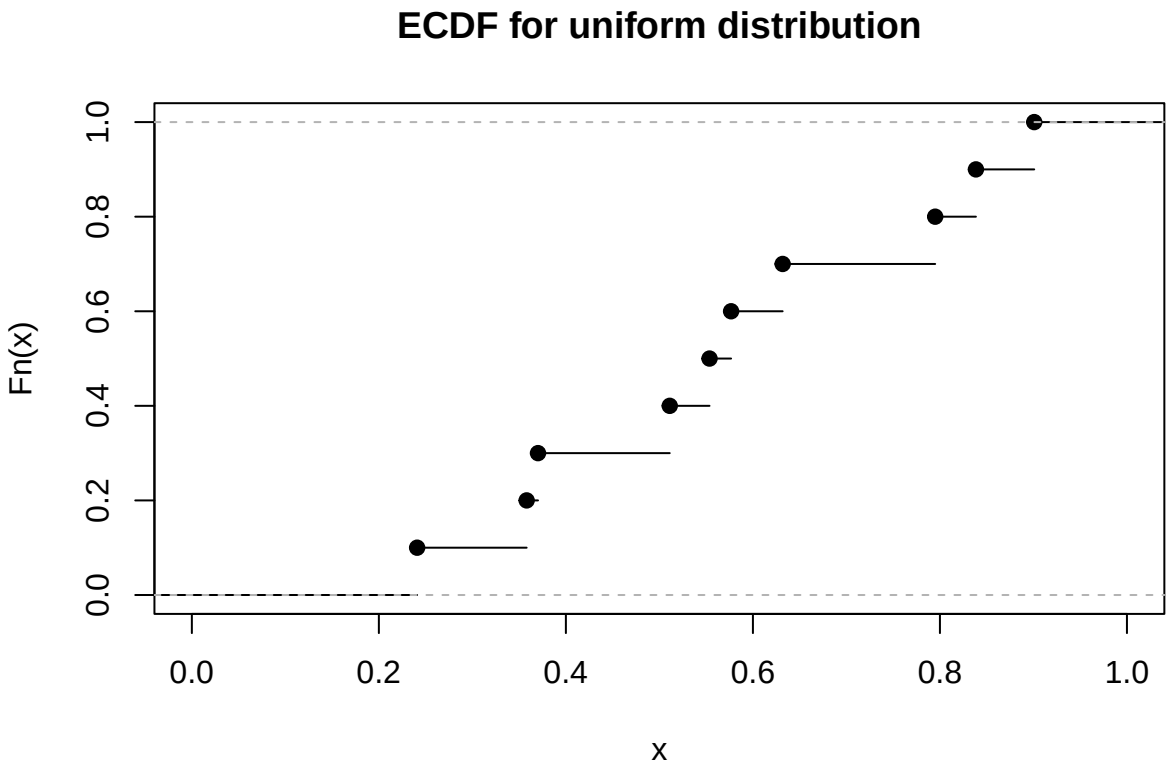
Bootstrap, originated from Efron (1979), is an extremely powerful and influential idea for estimation and inference. Here we give a brief introduction. Textbook exposition can be found in Cameron and Trivedi (2005) (Chapter 11).

Let $X_1, X_2, \ldots, X_n \sim F$ be an i.i.d. sample of n observations following a distribution F. The finite sample distribution of a statistic $T_n(\theta) \sim G_n(\cdot, F)$ usually depends on the sample size n, as well as the known true distribution F. Asymptotic theory approximates $G_n(\cdot, F)$ by its limit

$$G(\cdot, F) := \lim_{n\to\infty} G_n(\cdot, F).$$

In particular, if $T_n(\theta)$ is *asymptotically pivotal*, then $G_n(\cdot, F)$ is independent of F and it becomes $G(\cdot)$.

```
runif(10) %>%
  ecdf() %>%
  plot(, xlim = c(0, 1), main = "ECDF for uniform distribution")
```



Instead of referring to the limiting distribution, Bootstrap replaces the unknown distribution F in $G_n(\cdot, F)$ by a consistent estimator F_n of the true distribution, for example, the empirical distribution function $\hat{F}_n(\cdot) = n^{-1} \sum_{i=1}^{n} 1\{\cdot \leq X_i\}$. Bootstrap inference is drawn from the bootstrap distribution

$$G_n^*(\cdot) := G_n(\cdot, \hat{F}_n)$$

Implementation of bootstrap is a simulation exercise. In an i.i.d. environment we sample over each observation with equal weight, which is called *nonparametric bootstrap*. However, the species of bootstrap creatures has many varieties adapted to their living environment. In a dependent dataset such as time series (Chang 2004), clustering data or networks, we must adjust the sampling schedule to preserve the dependence structure. In regression context if we hold the independent variables fixed and only bootstrap the residual, we call it *parametric bootstrap*. If the error term is heteroskedascity, the relationship between X and $\hat{e}$ can be preserved by *wild bootstrap* (Davidson and MacKinnon 2010).

```
n <- 9 # sample size
boot_Rep <- 3 # bootstrap 3 times

real_sample <- rnorm(n) # the real sample
d0 <- tibble(no = 1:n, x = real_sample)
print(d0)
```

```
## # A tibble: 9 x 2
##      no      x
##   <int>  <dbl>
## 1     1 -0.855
## 2     2 -1.20
## 3     3 -0.335
## 4     4  0.366
## 5     5 -1.97
## 6     6  0.629
## 7     7 -1.18
## 8     8  1.96
## 9     9 -0.801
```

```
d_boot <- list() # save the bootstrap sample
for (b in 1:boot_Rep) {
  boot_index <- sample(1:n, n, replace = TRUE)
  d_boot[[b]] <- tibble(no = boot_index, x = real_sample[boot_index])
}

d_boot %>% as_tibble(, .name_repair = "minimal") %>% print()
```

```
## # A tibble: 9 x 3
##   ``$no     $x ``$no     $x ``$no     $x
##   <int>  <dbl> <int>  <dbl> <int>  <dbl>
## 1     6  0.629     4  0.366     5 -1.97
## 2     4  0.366     2 -1.20      7 -1.18
## 3     3 -0.335     5 -1.97      4  0.366
## 4     2 -1.20      7 -1.18      4  0.366
## 5     4  0.366     4  0.366     1 -0.855
## 6     4  0.366     3 -0.335     5 -1.97
## 7     4  0.366     9 -0.801     8  1.96
## 8     5 -1.97      8  1.96      8  1.96
## 9     6  0.629     1 -0.855     2 -1.20
```

In many regular cases, it is possible to show in theory the *consistency* of bootstrap: the statistic of interest and its bootstrap version converge to the same asymptotic distribution, or $G_n^*(a) \xrightarrow{p} G(a)$ for a such that $G(a)$ is continuous. However, bootstrap consistency can fail when the distribution of the statistic is discontinuous in the limit. Bootstrap is invalid in such cases. For instance, na茂ve bootstrap fails to replicate the asymptotic distribution of the two-stage least squares estimator under weak instruments. More sophisticated alternatives are need to fix the inconsistency of bootstrap, which we don't mention in this lecture.

4.2.1 Bootstrap Estimation

Bootstrap is simple enough to be done by a `ply`-family function for repeated simulations. Alternatively, R package boot provides a general function `boot()`.

Bootstrap is useful when the analytic formula of the variance of an econometric estimator is too complex to derive or code up.

Example

One of the most popular estimators for a sample selection model is Heckman (1977)'s two-step method. Let the outcome equation be

$$y_i = x_i\beta + u_i$$

and the selection equation be

$$D_i = z_i\gamma + v_i$$

To obtain a point estimator, we simply run a Probit in the selection model, predict the probability of participation, and then run an OLS of y_i on x_i and $\lambda(\hat{D}_i)$ in the outcome model, where $\lambda(\cdot)$ is the inverse Mill's ratio. However, as we can see from Heckman (1979)'s original paper, the asymptotic variance expression of the two-step estimator is very complicated. Instead of following the analytic formula, we can simply bootstrap the variance.

```
# the dataset comes from
# Greene( 2003 ): example 22.8, page 786
library(sampleSelection)
data(Mroz87)
# equations
selection_eq <- lfp ~ -1 + age + faminc + exper + educ
outcome_eq <- wage ~ exper + educ

# Heckman two-step estimation
heck <- sampleSelection::heckit(selection_eq, outcome_eq, data = Mroz87)
print(lmtest::coeftest(heck))
```

```
##
## z test of coefficients:
##
##                  Estimate  Std. Error z value  Pr(>|z|)
## age           -3.8037e-02  4.7335e-03 -8.0357 9.306e-16 ***
```

```
## faminc          1.0433e-05  4.3179e-06  2.4163   0.01568 *
## exper           7.4747e-02  7.1535e-03 10.4491 < 2.2e-16 ***
## educ            6.3923e-02  1.6283e-02  3.9257 8.646e-05 ***
## (Intercept)    -1.9611e+00  1.7375e+00 -1.1287   0.25904
## exper           1.6135e-02  3.3126e-02  0.4871   0.62619
## educ            4.8222e-01  8.0096e-02  6.0205 1.739e-09 ***
## invMillsRatio -3.0237e-01  9.6202e-01 -0.3143   0.75328
## sigma           3.1080e+00          NA      NA        NA
## rho            -9.7290e-02          NA      NA        NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Below is the function for a single bootstrap. For convenience, we save the point estimates of `heckit` but ignore the estimated variance. (This is a lazy option though, it must produce the same result if we take the effort to manually program the Heckit point estimation.) Implementation is just a repeated evaluation.

```
n <- nrow(Mroz87)
boot_heck <- function() {
  indices <- sample(1:n, n, replace = T) # resample the index set
  Mroz87_b <- Mroz87[indices, ] # generate the bootstrap sample
  heck_b <- sampleSelection::heckit(selection_eq, outcome_eq, data = Mroz87_b)
  return(coef(heck_b))
}
# repeat the bootstrap
boot_Rep <- 199
Heck_B <- plyr::ldply(.data = 1:boot_Rep, .fun = function(i) boot_heck())

# collect the bootstrap outcomes
Heck_b_sd <- apply(Heck_B, 2, sd)[1:7]
print(Heck_b_sd)
```

```
##          age       faminc        exper         educ  (Intercept)        exper
## 4.683937e-03 4.844919e-06 7.959621e-03 1.790120e-02 2.345574e+00 4.082622e-02
##         educ
## 9.399000e-02
```

The standard errors from the analytical expression and those from bootstrap are comparable. Both are asymptotically consistent. The bootstrap estimates can also be used to directly compute the confidence intervals.

4.2.2 Bootstrap Test

Bootstrap is particularly helpful in inference. Indeed, we can rigorously prove that bootstrap enjoys high-order improvement relative to analytic asymptotic approximation if the test statistic is asymptotically pivotal (Hall and Horowitz 1996). Loosely speaking, if the test statistic is asymptotically pivotal, a bootstrap hypothesis testing can be more accurate than its analytic asymptotic counterpart.

Example

We use bootstrap to test a hypothesis about the population mean. The test is carried out by a t-statistic. The distribution of the sample is either *normal* or *zero-centered chi-square*. It will show that the bootstrap test size is more precise than that of the asymptotic approximation.

We first prepare the workhorse functions.

```
# the t-statistic for a null hypothesis mu
T_stat <- function(Y, mu) sqrt(n) * (mean(Y) - mu) / sd(Y)

# the bootstrap function
boot_test <- function(Y, boot_Rep) {
  # INPUT
  # Y: the sample
  # boot_Rep: number of bootstrap replications

  n <- length(Y)
  boot_T <- rep(0, boot_Rep)

  # bootstrap in action
  for (r in 1:boot_Rep) {
    indices <- sample.int(n, n, replace = T) # resampling the index
    resampled_Y <- Y[indices] # construct a bootstrap artificial sample
    boot_T[r] <- abs(T_stat(resampled_Y, mean(Y)))
    # the bootstrapped t-statistic
    # mu is replaced by "mean(Y)" to mimic the situation under the null
  }

  # bootstrap critical value
  boot_critical_value <- quantile(boot_T, 1 - alpha)
  # bootstrap test decision
  return(abs(T_stat(Y, mu)) > boot_critical_value)
}
```

A key point for bootstrap test is that the null hypothesis must be imposed no matter the hypothesized parameter is true value or not. Therefore the bootstrap t-statistic is

$$T_n^* = \frac{\bar{X}^* - \bar{X}}{s^*/\sqrt{n}}.$$

That is, the bootstrap t-statistic is centered at $\bar{X}$, the sample mean of F_n, rather than θ, the population mean of F. This is because in the bootstrap world the "true'' distribution is F_n. If we wrongly center the bootstrap t-statistic at θ, then the test will have no power when the null hypothesis is false.

The following chuck of code report the rejection probability from three decision rules.

```
compare <- function() {
  # this function generates a sample of n observations
```

```r
  # and it returns the testing results from three decision rules

  if (distribution == "normal") {
    X <- rnorm(n)
  }
  else if (distribution == "chisq") {
    X <- rchisq(n, df = 3) - 3
  }

  t_value_X <- T_stat(X, mu) # T-statistic

  # compare it to the 97.5% of t-distribution
  exact <- abs(t_value_X) > qt(0.975, df = n - 1)
  # compare it to the 97.5% of normal distribution
  asym <- abs(t_value_X) > 1.96
  # decision from bootstrap
  boot_rule <- boot_test(X, boot_Rep)

  return(c(exact, asym, boot_rule))
}

# set the parameters
n <- 20
distribution <- "normal"
boot_Rep <- 199
MC_rep <- 2000
alpha <- 0.05
mu <- 0

# Monte Carlo simulation and report the rejection probability
res <- plyr::ldply(.data = 1:MC_rep, .fun = function(i) compare())
colnames(res) <- c("exact", "asym", "bootstrap")
print(colMeans(res))
```

```
##     exact      asym bootstrap
##    0.0445    0.0620    0.0490
```

Here the nominal size of the test is 5%. The program reports the empirical size —the ratio between the number of rejections to the total number of replications. The closer is the empirical size to the nominal size, the more accurate is the test. We find here the bootstrap test is more accurate than the asymptotic test.

When the underlying distribution is a χ^2, the exact distribution is difficult to derive analytically. However, we can still compare the asymptotic size with the bootstrap size.

```r
distribution <- "chisq"

res <- plyr::ldply(.data = 1:MC_rep, .fun = function(i) compare())
```

```
colnames(res) <- c("exact?", "asym", "bootstrap")
print(colMeans(res))
```

```
##    exact?      asym bootstrap
##    0.0685    0.0835    0.0625
```

Again, here the "exact test" is no longer exact. The asymptotic test works fairly reasonable, while the bootstrap is closer to the nominal size 5%.

4.3 Reading

Efron and Hastie: Ch 10 and 11

4.4 Reference

5 Numerical Optimization

Except for a few Bayesian estimators, almost all estimators in econometrics, such as OLS, MLE, 2SLS, and GMM, are optimizers of some criterion functions. Understanding how to construct an optimization problem and how to implement optimization by oneself is the key step to transform from a consumer of econometrics to a developer of econometrics. Unfortunately, traditionally econometrics curriculum does not pay enough attention in numerical optimization. The consequence is that many students rely on the procedures that the econometric packages offer. They are unable to tailor econometric methods for their purposes; instead, they modify their data to meet standard econometric methods.

A general optimization problem is formulated as

$$\min_{\theta \in \Theta} f(\theta) \quad \text{s.t. } g(\theta) = 0, h(\theta) \leq 0,$$

where $f(\cdot) \in \mathbb{R}$ is a scalar-valued criterion function, $g(\theta) = 0$ is a vector of equality constraints, and $h(\theta) \leq 0$ is a vector of inequality constraints.

Most established numerical optimization algorithms aim at finding a local minimum. However, there is little guarantee that these methods should locate the global minimum when multiple local minima exist.

Optimization without the equality and/or inequality constraints is called an *unconstrained* problem; otherwise it is called a *constrained* problem. The constraints can be incorporated into the criterion function via Lagrangian. Economic students are very familiar with constrained optimization—consider utility maximization given a budget constraint.

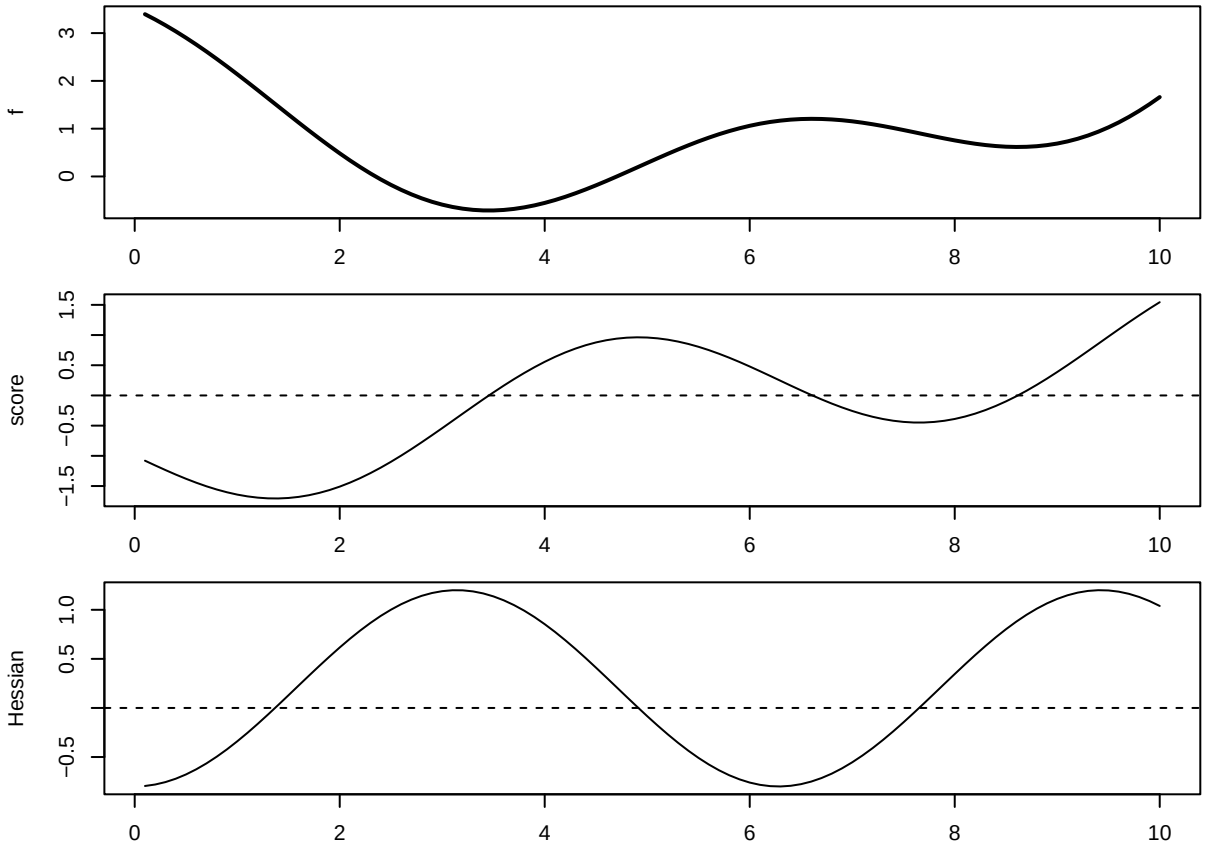
In terms of implementation, we always face the tradeoff between convenience and efficiency. Convenience is about the readability of the mathematical expressions and the code, while efficiency concerns the computing speed. We recommend that we put convenience as priority at the trial-and-error stage, and improves efficiency when necessary at a later stage for full-scale execution.

5.1 Methods

There are many optimization algorithms in the field of operational research; they are variants of a small handful of fundamental principles.

Many textbook MLE estimators are twice-differentiable but do not admit an explicit solution, for example Logit, Probit, and Tobit. The essential idea for optimizing a twice-differentiable objective function is the Newton's method. A necessary condition for optimization is the first-order condition $s(\theta) = \partial f(\theta)/\partial\theta = 0$.

```
f <- function(x) 0.1 * (x - 5)^2 + cos(x) # criterion
s <- function(x) 0.2 * (x - 5) - sin(x) # gradient
h <- function(x) 0.2 - cos(x) # Hessian

# plot
par(mfrow = c(3, 1))
par(mar = c(2, 4, 1, 2))

x_base <- seq(0.1, 10, by = 0.1)
plot(y = f(x_base), x = x_base, type = "l", lwd = 2, ylab = "f")
plot(y = s(x_base), x = x_base, type = "l", ylab = "score")
abline(h = 0, lty = 2)
plot(y = h(x_base), x = x_base, type = "l", ylab = "Hessian")
abline(h = 0, lty = 2)
```



At an initial trial value θ_0, if $s(\theta_0) \neq 0$, the search is updated by

$$\theta_{t+1} = \theta_t - (H(\theta_t))^{-1} s(\theta_t)$$

for the index of iteration $t = 0, 1, \cdots$, where $H(\theta) = \frac{\partial s(\theta)}{\partial \theta}$ is the Hessian matrix. This formulate can be intuitively motivated from a Taylor expansion at θ_t round $\theta_\star$, a root of $s(\cdot)$. Because $\theta_\star$ is a root,

$$0 = s(\theta_\star) = s(\theta_t) + H(\theta_t)(\theta_{t+1} - \theta_t) + O((\theta_{t+1} - \theta_t)^2).$$

Ignore the high-order term and rearrange, $\theta_\star = \theta_t - (H(\theta_t))^{-1} s(\theta_t)$, and we obtain the iteration formula by replacing $\theta_\star$ with the updated θ_{t+1}. In other words, it is a first-order linear updating formula for a nonlinear $s(\cdot)$. The algorithm iterates until $|\theta_{t+1} - \theta_t| < \epsilon$ (absolute criterion) and/or $|\theta_{t+1} - \theta_t|/|\theta_t| < \epsilon$ (relative criterion), where ϵ is a small positive number chosen as a tolerance level.

```
# Newton's method
Newton <- function(x) {
  x - s(x) / h(x)
} # update formula

x_init <- 6 # can experiment with various initial values

gap <- 1
epsilon <- 0.0001 # tolerance
while (gap > epsilon) {
```

```
  x_new <- Newton(x_init) %>% print()
  gap <- abs(x_init - x_new)
  x_init <- x_new
}
```

```
## [1] 6.630669
## [1] 6.611217
## [1] 6.611302
```

Newton's Method. Newton's method seeks the solution to $s(\theta) = 0$. Recall that the first-order condition is a necessary condition but not a sufficient condition. We need to verify the second-order condition for each root of $s(\theta)$ to decide whether it is a minimizer, maximizer or saddle point. If there are multiple minima, we compare the value at each to decide the global minimum.

It is clear that Newton's method requires computing the gradient $s(\theta)$ and the Hessian $H(\theta)$. Newton's method numerically converges at quadratic rate.

Quasi-Newton Method. The most well-known quasi-Newton algorithm is BFGS. It avoids explicit calculation of the computationally expensive Hessian matrix. Instead, starting from an initial (inverse) Hessian, it updates the Hessian by an explicit formula motivated from the idea of quadratic approximation.

Derivative-Free Method. Nelder-Mead is a simplex method. It searches a local minimum by reflection, expansion and contraction.

5.2 Implementation

R's optimization infrastructure has been constantly improving. R Optimization Task View gives a survey of the available CRAN packages. For general-purpose nonlinear optimization, the package `optimx` (Nash 2014) effectively replaces R's default optimization commands. `optimx` provides a unified interface for various widely-used optimization algorithms. Moreover, it facilitates comparison among optimization algorithms. A relatively new package `ROI` (Theu雅l, Schwendinger, and Hornik 2019) attempts to offer a consistent modeling framework to communicate with solvers. We will incorporate `ROI` in a future draft.

Example

We use `optimx` to solve pseudo Poisson maximum likelihood estimation (PPML), which is a popular estimator in international trade for cross-country bilateral trade (Silva and Tenreyro 2006). If y_i is a continuous random variable, it obviously does not follow a Poisson distribution whose support consists of non-negative integers. However, if the conditional mean model

$$E[y_i|x_i] = \exp(x_i'\beta),$$

is satisfied, we can still use the Poisson regression to obtain a consistent estimator of the parameter β even if y_i does not follow a conditional Poisson distribution.

If Z follows a Poisson distribution with mean λ, the probability mass function

$$\Pr(Z = k) = \frac{\mathrm{e}^{-\lambda}\lambda^k}{k!}, \text{ for } k = 0, 1, 2, \ldots,$$

so that

$$\log \Pr(Y = y|x) = -\exp(x'\beta) + y \cdot x'\beta - \log k!$$

Since the last term is irrelevant to the parameter, the log-likelihood function is

$$\ell(\beta) = \log \Pr(\mathbf{y}|\mathbf{x}; \beta) = -\sum_{i=1}^{n} \exp(x_i'\beta) + \sum_{i=1}^{n} y_i x_i'\beta.$$

In addition, it is easy to write the gradient

$$s(\beta) = \frac{\partial \ell(\beta)}{\partial \beta} = -\sum_{i=1}^{n} \exp(x_i'\beta)x_i + \sum_{i=1}^{n} y_i x_i.$$

and verify that the Hessian

$$H(\beta) = \frac{\partial^2 \ell(\beta)}{\partial \beta \partial \beta'} = -\sum_{i=1}^{n} \exp(x_i'\beta)x_i x_i'$$

is negative definite. Therefore, $\ell(\beta)$ is strictly concave in β.

In operational research, the default optimization is minimization, although utility is maximized in economics and likelihood is maximized in statistics. To follow this convention in operational research, here we formulate the *negative* log-likelihood.

```
# Poisson likelihood
poisson.loglik <- function(b) {
  b <- as.matrix(b)
  lambda <- exp(X %*% b)
  ell <- -sum(-lambda + y * log(lambda))
  return(ell)
}
```

To implement optimization in `R`, it is recommended to write the criterion as a function of the parameter. Data can be fed inside or outside of the function. If the data is provided as additional arguments, these arguments must be explicit. (In contrast, in `Matlab` the parameter must be the sole argument for the function to be optimized, and data can only be injected through a nested function.)

```
# implement both BFGS and Nelder-Mead for comparison.

library(AER)

## prepare the data
data("RecreationDemand")
y <- RecreationDemand$trips
X <- with(RecreationDemand, cbind(1, income))
```

```r
## estimation
b.init <- c(0, 1) # initial value
b.hat <- optimx::optimx(b.init, poisson.loglik,
  method = c("BFGS", "Nelder-Mead"),
  control = list(
    reltol = 1e-7,
    abstol = 1e-7
  )
)
print(b.hat)
```

```
##                    p1          p2    value fevals gevals niter convcode  kkt1
## BFGS        1.177411 -0.09994234 261.1141     99     21    NA         0  TRUE
## Nelder-Mead 1.167261 -0.09703975 261.1317     53     NA    NA         0 FALSE
##             kkt2 xtime
## BFGS        TRUE  0.03
## Nelder-Mead TRUE  0.00
```

Given the conditional mean model, nonlinear least squares (NLS) is also consistent in theory. NLS minimizes

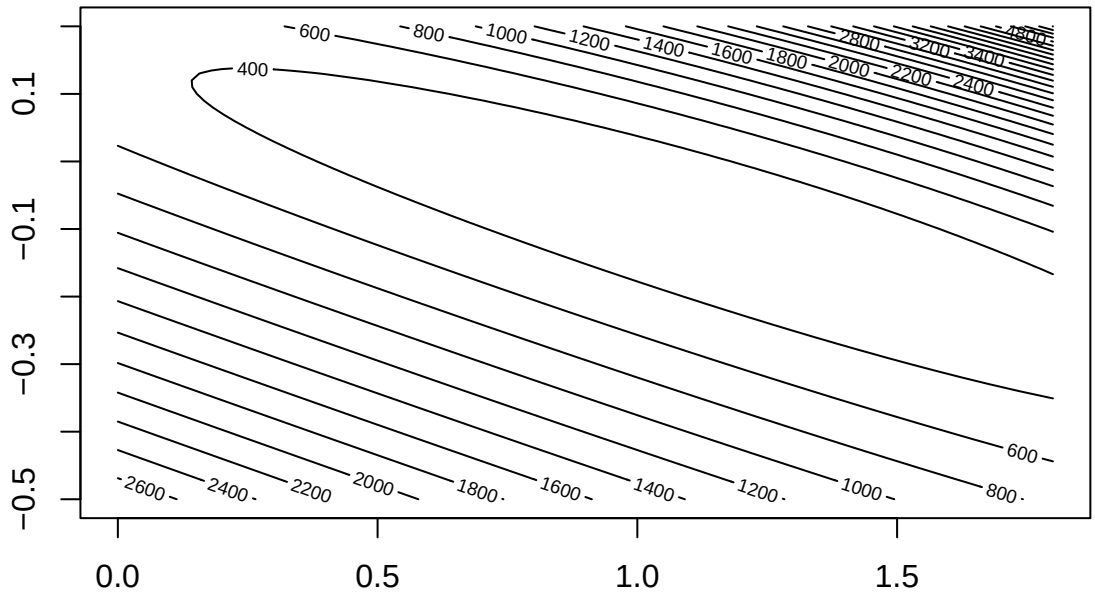
$$\sum_{i=1}^{n}(y_i - \exp(x_i\beta))^2$$

A natural question is: why do we prefer PPML to NLS? PPML's optimization for the linear index is globally convex, while NLS is not. It implies that the numerical optimization of PPML is easier and more robust than that of NLS. I leave the derivation of the non-convexity of NLS as an exercise.

In practice no algorithm suits all problems. Simulation, where the true parameter is known, is helpful to check the accuracy of one's optimization routine before applying to an empirical problem, where the true parameter is unknown. Contour plot is a useful tool to visualize the function surface/manifold in a low dimension.

Example

```r
x.grid <- seq(0, 1.8, 0.02)
x.length <- length(x.grid)
y.grid <- seq(-.5, .2, 0.01)
y.length <- length(y.grid)

z.contour <- matrix(0, nrow = x.length, ncol = y.length)

for (i in 1:x.length) {
  for (j in 1:y.length) {
    z.contour[i, j] <- poisson.loglik(c(x.grid[i], y.grid[j]))
  }
}

contour(x.grid, y.grid, z.contour, 20)
```



For problems that demand more accuracy, third-party standalone solvers can be invoked via interfaces to `R`. For example, we can access `NLopt` through the packages `nloptr`.

`NLopt` offers an extensive list of algorithms.

Example

We first carry out the Nelder-Mead algorithm in NLOPT.

```
## optimization with NLoptr

opts <- list(
  "algorithm" = "NLOPT_LN_NELDERMEAD",
  "xtol_rel" = 1.0e-7,
  maxeval = 500
)

res_NM <- nloptr::nloptr(
  x0 = b.init,
  eval_f = poisson.loglik,
  opts = opts
)
print(res_NM)
```

```
##
```

```
## Call:
## nloptr::nloptr(x0 = b.init, eval_f = poisson.loglik, opts = opts)
##
##
## Minimization using NLopt version 2.7.1
##
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 114
## Termination conditions:  xtol_rel: 1e-07 maxeval: 500
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  261.114078295329
## Optimal value of controls: 1.177397 -0.09993984
```

```
# "SLSQP" is indeed the BFGS algorithm in NLopt,
# though "BFGS" doesn't appear in the name
opts <- list("algorithm" = "NLOPT_LD_SLSQP", "xtol_rel" = 1.0e-7)
```

To invoke BFGS in NLOPT, we must code up the gradient $s(\beta)$, as in the function `poisson.log.grad()` below.

```
poisson.loglik.grad <- function(b) {
  b <- as.matrix(b)
  lambda <- exp(X %*% b)
  ell <- -colSums(-as.vector(lambda) * X + y * X)
  return(ell)
}
```

We compare the analytical gradient with the numerical gradient to make sure the function is correct.

```
# check the numerical gradient and the analytical gradient
b <- c(0, .5)
numDeriv::grad(poisson.loglik, b)
```

```
## [1]  6542.46 45825.40
```

```
poisson.loglik.grad(b)
```

```
##            income
##  6542.46 45825.40
```

With the function of gradient, we are ready for BFGS.

```
res_BFGS <- nloptr::nloptr(
  x0 = b.init,
  eval_f = poisson.loglik,
  eval_grad_f = poisson.loglik.grad,
  opts = opts
)
```

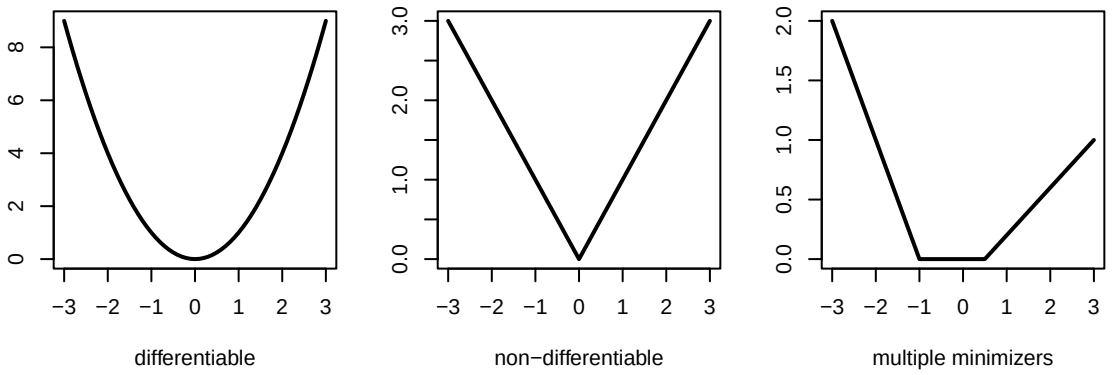
```
print(res_BFGS)
```

```
##
## Call:
##
## nloptr::nloptr(x0 = b.init, eval_f = poisson.loglik, eval_grad_f = poisson.loglik.grad,
##     opts = opts)
##
##
## Minimization using NLopt version 2.7.1
##
## NLopt solver status: 4 ( NLOPT_XTOL_REACHED: Optimization stopped because
## xtol_rel or xtol_abs (above) was reached. )
##
## Number of Iterations....: 41
## Termination conditions:  xtol_rel: 1e-07
## Number of inequality constraints:  0
## Number of equality constraints:    0
## Optimal value of objective function:  261.114078295329
## Optimal value of controls: 1.177397 -0.09993984
```

5.3 Convex Optimization

If a function is convex in its argument, then a local minimum is a global minimum. Convex optimization is particularly important in high-dimensional problems. The readers are referred to Boyd and Vandenberghe (2004) for an accessible comprehensive treatment. They claim that "convex optimization is technology; all other optimizations are arts." This is true to some extent.

```
f1 <- function(x) x^2
f2 <- function(x) abs(x)
f3 <- function(x) (-x - 1) * (x <= -1) + (0.4 * x - .2) * (x >= .5)

par(mfrow = c(1, 3))
par(mar = c(4, 2, 1, 2))

x_base <- seq(-3, 3, by = 0.1)
plot(y = f1(x_base), x = x_base, type = "l", lwd = 2, xlab = "differentiable")
plot(y = f2(x_base), x = x_base, type = "l", lwd = 2, xlab = "non-differentiable")
plot(y = f3(x_base), x = x_base, type = "l", lwd = 2, xlab = "multiple minimizers")
```



Example

- linear regression model MLE

Normal MLE. The (negative) log-likelihood is

$$\ell(\beta, \sigma) = \log \sigma + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i'\beta)^2$$

This is not a convex problem, because $\log \sigma$ is concave. But if we re-parameterize the criterion function by $\gamma = 1/\sigma$ and $\alpha = \beta/\sigma$, then

$$\ell(\alpha, \gamma) = -\log \gamma + \frac{1}{2} \sum_{i=1}^{n} (\gamma y_i - x_i'\alpha)^2$$

in convex in α, γ. Many MLE estimators in econometric textbooks are convex.

In view of the importance of high-dimensional estimation problems, (**gao2018two?**) explore the infrastructure in R to carry out convex optimization with two econometric examples. There are several options. `CVXR` (Fu, Narasimhan, and Boyd 2019) is a convenient convex modeling language that supports proprietary convex solvers `CLEPX`, `MOSEK`, `Gurubi` as well as open-source solvers `ECOS` and `SDPT3`. While open-source solvers does not require license and can be installed in large scale in cloud computing, proprietary solvers are more faster and more reliable. `MOSEK` offers free academic license and we have had extensive experience with it. `Rmosek` offers an interface in `R` to access `Mosek` (`Rtools` is a prerequisite to install `Rmosek` in Windows.)

Example: Relaxed empirical likelihood

Consider a model with a "true" parameter β_0 satisfying the moment condition $\mathrm{E}\left[h\left(Z_i, \beta_0\right)\right] = 0_m$, where $\{Z_i\}_{i=1}^n$ is the observed data, β is a low dimensional parameter of interest, and h is an $\mathbb{R}^m$-valued moment function. Empirical likelihood (EL) (Owen 1988) (Qin and Lawless 1994) solves

$$\max_{\beta \in \mathcal{B}, \pi \Delta_n} \sum_{i=1}^{n} \log \pi_i \quad \text{s.t.} \quad \sum_{i=1}^{n} \pi_i h\left(Z_i, \beta\right) = 0_m$$

where $\Delta_n = \{\pi \in [0,1]^n : \sum_{i=1}^n \pi_i = 1\}$ is the n-dimensional probability simplex.

To handle the high-dimensional case, i.e., $m > n$, Shi (2016a) proposes the relaxed empirical

likelihood (REL), defined as the solution to

$$\max_{\beta\in\mathcal{B}}\max_{\pi\in\Delta_n^\lambda(\beta)}\sum_{i=1}^n\log\pi_i$$

where

$$\Delta_n^\lambda(\beta)=\left\{\pi_i\in\Delta_n:\left|\sum_{i=1}^n\pi_i h_{ij}(\beta)\right|\le\lambda,\ j=1,2,\cdots,m\right\}$$

is a relaxed simplex, $\lambda\ge 0$ is a tuning parameter, $h_{ij}(\beta)=h_j(Z_i,\beta)$ is the j-th component of $h(Z_i,\beta)$.

Similar to standard EL, REL's optimization involves an inner loop and an outer loop. The outer loop for β is a general low-dimensional nonlinear optimization, which can be solved by Newton-type methods. With the linear constraints and the logarithm objective, the inner loop is convex in $\pi=(\pi_i)_{i=1}^n$. By introducing auxiliary variable, t_i, the logarithm objective can be formulated as a linear objective function $\sum_{i=1}^n t_i$ and n exponential conic constraints, $(\pi_i,1,t_i)\in\mathcal{K}_{\exp}=\{(x_1,x_2,x_3):x_1\ge x_2\exp(x_3/x_2),x_2>0\}\cup\{(x_1,0,x_3):x_1\ge 0,x_3\le 0\}$, $i=1,2,\cdots,n$.

For each β, the inner problem can be then formulated as a conic programming problem,

$$\begin{aligned}&\max_{\pi,t}\sum_{i=1}^n t_i\\ \text{s.t. }&\begin{bmatrix}1\\-\lambda\\\vdots\\-\lambda\end{bmatrix}\le\begin{bmatrix}1&1&\cdots&1\\h_{11}(\beta)&h_{21}(\beta)&\cdots&h_{n1}(\beta)\\\vdots&\vdots&\ddots&\vdots\\h_{1m}(\beta)&h_{2m}(\beta)&\cdots&h_{nm}(\beta)\end{bmatrix}\begin{bmatrix}\pi_1\\\pi_2\\\vdots\\\pi_n\end{bmatrix}\le\begin{bmatrix}1\\\lambda\\\vdots\\\lambda\end{bmatrix}\\&(\pi_i,1,t_i)\in\mathcal{K}_{\exp},0\le\pi_i\le 1,\ \text{for each } i=1,2,\cdots,n\end{aligned}$$

To understand the exponential cone, notice that $(\pi_i,1,t_i)\in\mathcal{K}_{\exp}$ is equivalent to $\{\pi_i\ge\exp(t_i):\pi_i\ge 0,t_i\le 0\}$. It implies $t_i\le\log\pi_i$. Since the problem maximizes $\sum t_i$, we must have $t_i=\log\pi_i$. The constrained optimization is readily solvable in `Rmosek` by translating the mathematical expression into computer code.

```
innerloop <- function(b, y, X, Z, tau) {
  n <- nrow(Z)
  m <- ncol(Z)

  # Generate moment condition
  H <- MomentMatrix(y, X, Z, b)

  # Initialize the mosek problem
  Prob <- list(sense = "max")

  # Prob$dparam$intpnt_nl_tol_rel_gap <- 1e-5;
  Prob$dparam <- list(INTPNT_CO_TOL_REL_GAP = 1e-5)

  # Linear coefficients of the objective
```

```
  Prob$c <- c(rep(0, n), rep(1, n), rep(0, n))

  # Linear constraints
  H_tilde <- Matrix(rbind(rep(1, n), H), sparse = TRUE)
  A <-
    rbind(
      cbind(H_tilde, Matrix(0, m + 1, 2 * n, sparse = TRUE)),
      cbind(Matrix(0, n, 2 * n, sparse = TRUE), Diagonal(n))
    )
  Prob$A <- A
  Prob$bc <-
    rbind(c(1, rep(-tau, m), rep(1, n)), c(1, rep(tau, m), rep(1, n)))
  Prob$bx <- rbind(
    c(rep(0, n), rep(-Inf, n), rep(1, n)),
    c(rep(1, n), rep(0, n), rep(1, n))
  )

  # Exponential Cones
  NUMCONES <- n
  Prob$cones <- matrix(list(), nrow = 2, ncol = NUMCONES)
  rownames(Prob$cones) <- c("type", "sub")
  for (i in 1:n) {
    Prob$cones[, i] <- list("PEXP", c(i, 2 * n + i, n + i))
  }

  # Invoke Mosek
  mosek.out <- mosek(Prob, opts = list(verbose = 0, soldetail = 1))

  if (mosek.out$sol$itr$solsta == "OPTIMAL") {
    # Since the default of NLOPTR is to do minimization, need to set it as negative
    return(-mosek.out$sol$itr$pobjval)
  } else {
    warning("WARNING: Inner loop not optimized")
    return(Inf)
  }
}
```

The inner loop optimization can also be carried out by `CVXR`. This code snippet is shorter than easier to read.

```
innerloop.cvxr <- function(b, y = NULL, X = NULL, Z = NULL, tau = NULL) {
  n <- nrow(Z)
  m <- ncol(Z)

  H <- MomentMatrix(y, X, Z, b)

  p <- Variable(n)
  constr <- list(
```

```
    sum(p) == 1,
    p >= 0,
    p <= 1,
    H %*% p >= -tau,
    H %*% p <= tau
  )

  obj <- sum(log(p))
  obj <- Maximize(obj)

  Prob <- Problem(obj, constr)
  cvxr.out <- solve(Prob)

  if (cvxr.out$status == "optimal") {
    return(-cvxr.out$value)
  } else {
    warning("WARNING: Inner loop not optimized")
    return(Inf)
  }
}
```

5.4 Future writing plan

- more convex optimization examples, for example Lasso, portfolio optimization (Shi, Su, and Xie 2020)
- Add `ROI`.

5.5 Reading

- Fu, Narasimhan, and Boyd (2019)
- (**gao2018two?**)
- Theu雅l, Schwendinger, and Hornik (2019)

5.6 References

6 From Nonparametrics to Machine Learning

Machine learning has quickly grown into a big field, with applications from scientific research to daily life. An authoritative reference is Friedman, Hastie, and Tibshirani (2008), written at the entry-year postgraduate level. The ideas in machine learning are general and applicable to economic investigation. Athey (2018) discusses the impact of machine learning techniques to economic analysis. Mullainathan and Spiess (2017) survey a few new commonly used methods and demonstrate them in

a real example. Taddy (2018) introduces new technology *artificial intelligence* and the implication of the underlying economic modeling.

The two broad classes of machine learning methods are *supervised learning* and *unsupervised learning*. Roughly speaking, the former is about the connection between X and Y, while the latter is only about X. Instances of the former are various regression and classification methods; those of the latter are density estimation, principal component analysis, and clustering. These examples are all familiar econometric problems.

From an econometrician's view, supervised machine learning is a set of data fitting procedures that focus on out-of-sample prediction. The simplest illustration is in the regression context. We repeat a scientific experiment for n times, and we harvest a dataset $(y_i, x_i)_{i=1}^n$. What would be the best way to predict y_{n+1} from the same experiment if we know x_{n+1}?

Machine learning is a paradigm shift against conventional statistics. When a statistician propose a new estimator, the standard practice is to pursue three desirable properties one after another. We first establish its consistency, which is seen as the bottom line. Given consistency, we want to show its asymptotic distribution. Ideally, the asymptotic distribution is normal. Asymptotic normality is desirable as it holds for many regular estimators and the inferential procedure is familiar to applied researchers. Furthermore, for an asymptotically normal estimator, we want to show efficiency, an optimality property. An efficient estimator achieves the smallest asymptotic variance in a class of asymptotically normal estimators.

In addition, econometrician also cares about model identification and economic interpretation of the empirical results. Econometrics workflow interacts the data at hand and the model of interest. At the population level, we think about the problem of identification. Once the parameter of interest is identified, then we can proceed to parameter estimation and inference. Finally, we interpret the results and hopefully they shed light on economics.

Machine learning deviates from such routines. First, they argue efficiency is not crucial because the dataset itself is big enough so that the variance is usually small. Second, in many situations statistical inference is not the goal, so inferential procedure is not of interest. For example, the recommendation system on Amazon or Taobao has a machine learning algorithm behind it. There we care about the prediction accuracy, not the causal link why a consumer interested in one good is likely to purchase another good. Third, the world is so complex that we have little idea about how the data is generated. We do not have to assume a data generating process (DGP). If there is no DGP, we lose the standing ground to talk about consistency. Where would my estimator converge to if there is no "true parameter"? With these arguments, the paradigm of conventional statistics is smashed. In the context of econometrics, such argument completely rejects the structural modeling tradition (the Cowles approach).

Readers interested in the debate are referred to Breiman (2001b). In this lecture, we put aside the ongoing philosophical debate. Instead, we study the most popular machine learning methods that have found growing popularity in economics.

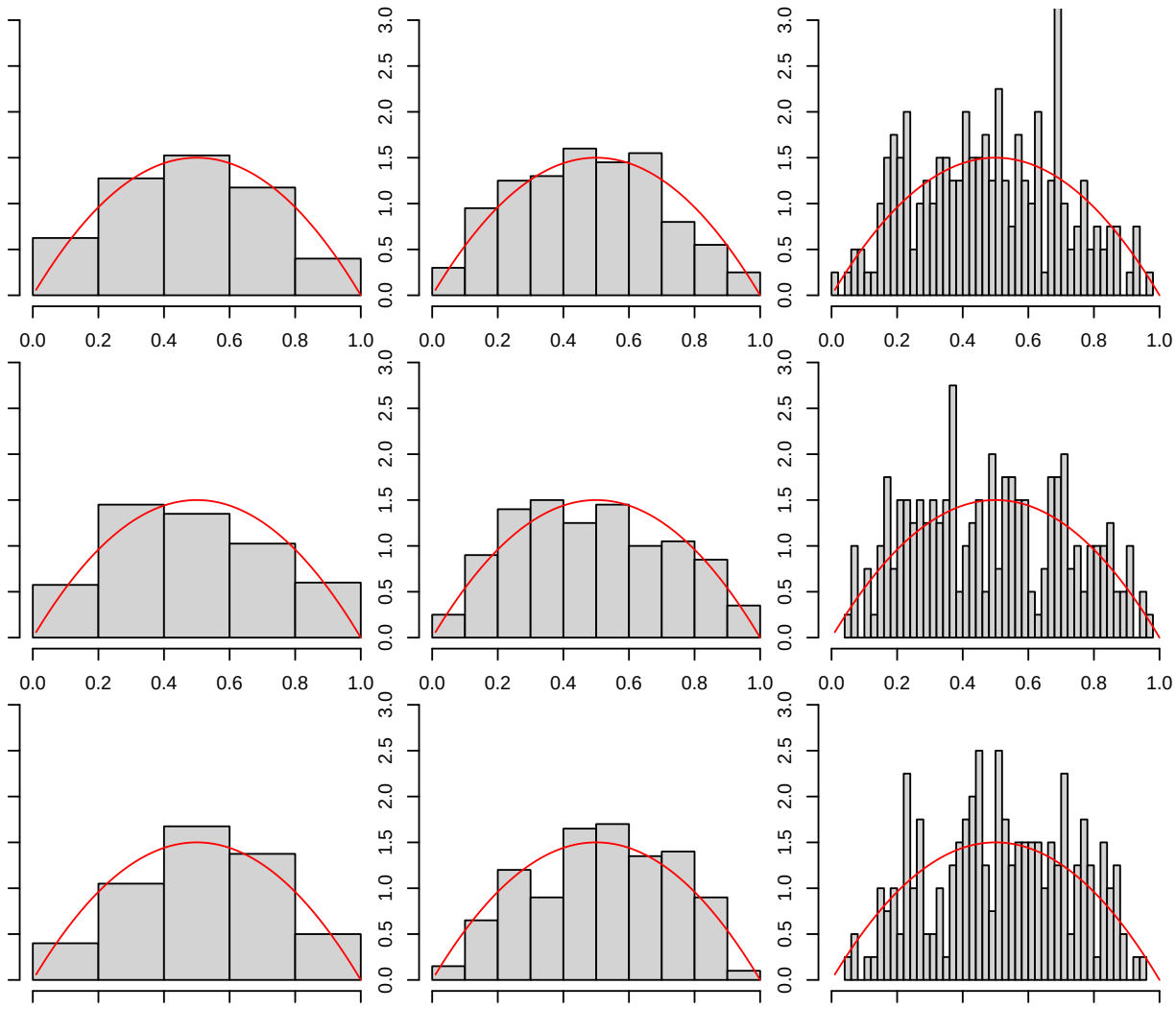
6.1 Nonparametric Estimation

Parametric is referred to problems with a finite number of parameters, whereas *nonparametric* is associated with an infinite number of parameters. Nonparametric estimation is nothing new to

statisticians. However, some ideas in this old topic is directly related to the underlying principles of machine learning methods.

Consider the density estimation given a sample $(x_1, \ldots, x_n)$. If we assume that the sample is drawn from a parametric family, for example the normal distribution, then we can use the maximum likelihood estimation to learn the mean and the variance. Nevertheless, when the parametric family is misspecified, the MLE estimation is inconsistent in theory, and we can at best identify a *pseudo true value*. In practice, what is the correct parametric family is unknown. If we do not want to impose a parametric assumption, then in principle we will have to use an infinite number of parameters to fully characterize the density. One well-known nonparametric estimation is the histogram. The shape of the bars of the histogram depends on the partition of the support. If the grid system on the support is too fine, then each bin will have only a few observations. Despite small bias, the estimation will suffer a large variance. On the other hand, if the grid system is too coarse, then each bin will be wide. It causes big bias, though the variance is small because each bin contains many observations. There is an bias-variance tradeoff. This tradeoff is the defining feature not only for nonparametric estimation but for all machine learning methods.

```
n <- 200

par(mfrow = c(3, 3))
par(mar = c(1, 1, 1, 1))

x_base <- seq(0.01,1,by = 0.01)
breaks_list = c(4, 12, 60)

for (ii in 1:3){
  x <- rbeta(n, 2, 2) # beta distribution
  for ( bb in breaks_list){
    hist(x, breaks = bb, main="", freq = FALSE, ylim = c(0,3),xlim = c(0,1))
    lines( y = dbeta( x_base, 2, 2), x = x_base , col = "red" )
  }
}
```



Another example of nonparametric estimation is the conditional mean $f(x) = E[y_i|x_i = x]$ given a sample (y_i, x_i). This is what we encountered in the first lecture of graduate econometrics Econ5121A. We solve the minimization problem

$$\min_f E[(y_i - f(x_i))^2]$$

In Econ5121A, we use the linear projection to approximate $f(x)$. But the conditional mean is in general a nonlinear function. If we do not know the underlying parametric estimation of (y_i, x_i), estimating $f(x)$ becomes a non-parametric problem. In practice, the sample size n is always finite. The sample minimization problem is

$$\min_f \sum_{i=1}^{n} (y_i - f(x_i))^2.$$

We still have to restrict the class of functions that we search for the minimizer. If we assume f is a continuous function, one way to estimate it is the kernel method based on density estimation.

An alternative is to use a series expansion to approximate the function. Series expansion generates

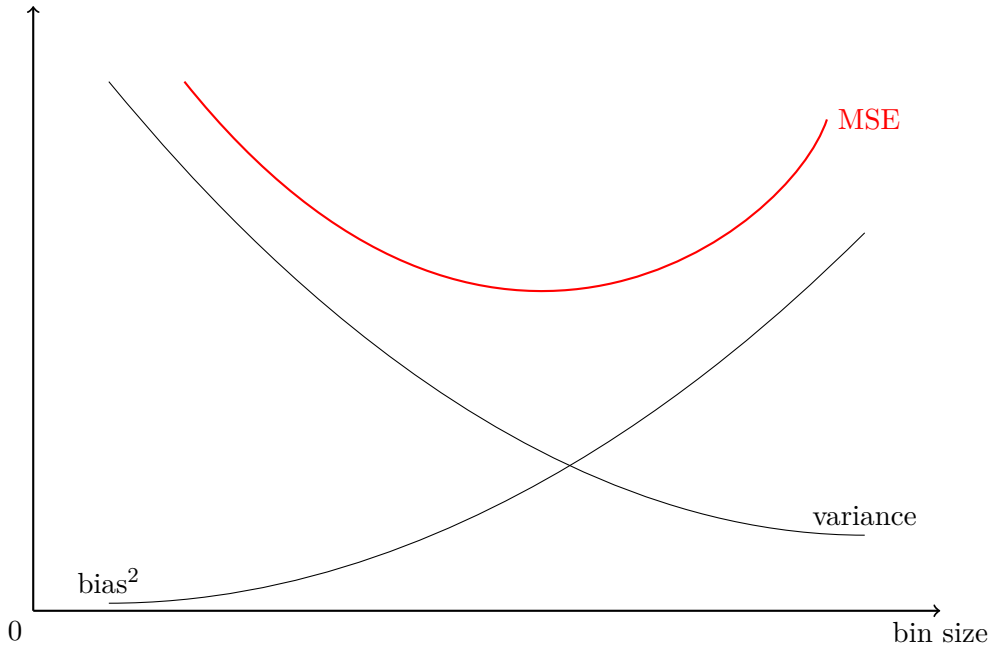


图 4: Bias-variance tradeoff

many additive regressors whose coefficients will be estimated. This is one way to "create" many variables on the right-hand side of a linear regression. For example, any bounded, continuous and differentiate function has a series representation $f(x) = \sum_{k=0}^{\infty} \beta_k \cos(\frac{k}{2}\pi x)$. In finite sample, we choose a finite K, usually much smaller than n, as a cut-off. Asymptotically $K \to \infty$ as $n \to \infty$ so that

$$f_K(x) = \sum_{k=0}^{K} \beta_k \cos\left(\frac{k}{2}\pi x\right) \to f(x).$$

Similar bias-variance tradeoff appears in this nonparametric regression. If K is too big, f will be too flexible and it can achieve 100% of in-sample R-squared. This is not useful for out-of-sample prediction. Such prediction will have large variance, but small bias. On the other extreme, a very small K will make $f_K(x)$ too rigid to approximate general nonlinear functions. It causes large bias but small variance.

The fundamental statistical mechanism that governs the performance is the bias-variance tradeoff. Thus we need *regularization* to balance the two components in the mean-squared error. Choosing the bandwidth is one way of regularization, choosing the terms of series expansion is another way of regularization.

A third way of regularization is to specify a sufficiently large K, and then add a penalty term to control the complexity of the additive series. The optimization problem is

$$\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \sum_{k=0}^{K} \beta_k f_k(x_i) \right)^2 + \lambda \sum_{k=0}^{K} \beta_k^2,$$

where λ is the tuning parameter such that $\lambda \to 0$ as $n \to \infty$, and $f_k(x_i) = \cos\left(\frac{k}{2}\pi x_i\right)$. In compact

notation, let $y = (y_1, \ldots, y_n)'$ and $X = (X_{ik} = f_k(x_i))$, the above problem can be written as

$$(2n)^{-1}(Y - X\beta)'(Y - X\beta) + \lambda\|\beta\|_2^2,$$

and this optimization has an explicit solution $\hat{\beta} = (X'X + \lambda I)^{-1}X'Y$. This is the *ridge regression* proposed in 1970's. This penalization scheme is very similar to what we will discuss in the next section in variable selection.

The practical question is, given a regularization problem, how to choose the tuning parameter? This is a difficult statistical problem with active research. The main theoretical proposal is either using an *information criterion* (for example, Akaike information criterion $\log\hat{\sigma}^2 + 2K$ or Bayesian information criterion $\log\hat{\sigma}^2 + K\log n$), or *cross validation*.

6.2 Data Splitting

The workflow of machine learning methods is quite different from econometrics. The main purpose is often prediction instead of interpretation. They use some "off-the-shelf" generic learning methods, and the models are measured by their performance in prediction. In order to avoid overfitting it is essential to tune at least a few tuning parameters.

Most machine learning methods take an agnostic view about the DGP, and they explicitly acknowledge model uncertainty. To address the issue of model selection (tuning parameter selection), in a data rich environment we split the data into three parts. A *training dataset* is used to fit the model parameter given the tuning parameters. A *validation dataset* is used to compare the out-of-sample performance under different tuning parameters. It helps decide a set of desirable tuning parameters. Ideally, the *testing sample* should be kept by a third party away from the modeler. The testing sample is the final judge of the relative merit of the fitted models.

The R package `caret` (Classification And REgression Training) provides a framework for many machine learning methods. The function `createDataPartition` splits the sample for both cross-sectional data and time series.

6.2.1 Cross Validation

An S-fold cross validation partitions the dataset into S disjoint sections. In each iteration, it picks one of the sections as the validation sample and the other $S-1$ sections as the training sample, and computes an out-of-sample goodness-of-fit measurement, for example *mean-squared prediction error* ${n_v}^{-1}\sum_{i\in val}(y_i - \hat{y}_i)^2$ where val is the validation set and n_v is its cardinality, or *mean-absolute prediction error* ${n_v}^{-1}\sum_{i\in val}|y_i - \hat{y}_i|$. Repeat this process for S times so that each of the S sections are treated as the validation sample, and average the goodness-of-fit measurement over the S sections to determined the best tuning parameter. If $S = n - 1$, it is called *leave-one-out cross validation*, but it can be computationally too expensive when n is big. Instead, in practice we can $S = 5$ for 10, called 5-fold cross validation or 10-fold cross validation, respectively.

In time series context, cross validation must preserve the dependence structure. If the time series is stationary, we can partition the data into S consecutive blocks. If the purpose is ahead-of-time forecasting, then we can use nested CV. The figure shows a nested CV with fixed-length rolling window scheme, while the sub-training data can also be an extending rolling window.

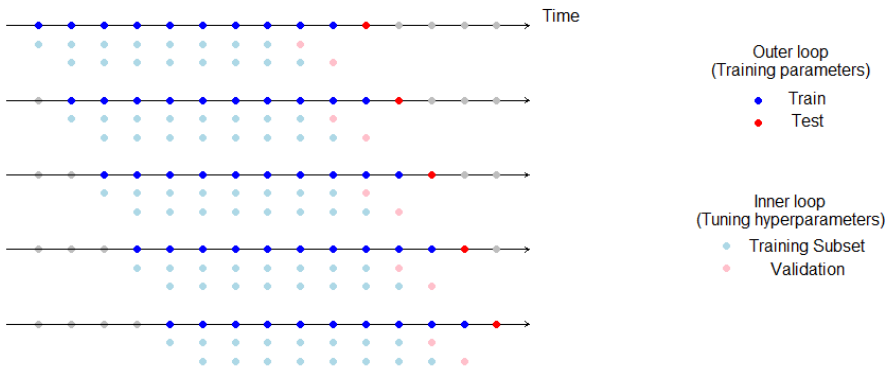


图 5: Rolling window time series cross validation

6.3 Variable Selection and Prediction

In modern scientific analysis, the number of covariates x_i can be enormous. In DNA microarray analysis, we look for association between a symptom and genes. Theory in biology indicates that only a small handful of genes are involved, but it does not pinpoint which ones are the culprits. Variable selection is useful to identify the relevant genes, and then we can think about how to edit the genes to prevent certain diseases and better people's life.

Explanatory variables are abundant in some empirical economic examples. For instance, a questionnaire from the UK Living Costs and Food Survey, a survey widely used for analysis of demand theory and family consumption, consists of thousand of questions. (**giannone2017economic?**) link experiment variable selection methods in 6 widely used economic datasets with many predictors.

Hazard of model selection To elaborate the distortion of test size when the t statistic is selected from two models in pursuit of significance.

$$\begin{pmatrix} y \\ x_1 \\ x_2 \end{pmatrix} \sim N\left(0, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \sqrt{0.5} \\ 0 & \sqrt{0.5} & 1 \end{pmatrix}\right)$$

Both x_1 and x_2 are independent of y. The test size dependens on the correlation between the two regressors. If the test is conducted for a single model, the size is the pre-specified 10%. If we try two models, the size is inflated to about 17%.

```
n <- 100
Rep <- 5000

t_stat <- function(y, x) {
  beta_hat <- sum(x * y) / sum(x^2)
  e_hat <- y - beta_hat * x
  sigma2_hat <- var(e_hat)
  t_stat <- beta_hat / sqrt(sigma2_hat / sum(x^2))
  return(t_stat)
}
```

```
res <- matrix(NA, Rep, 2)

for (r in 1:Rep) {
  y <- rnorm(n)
  x1 <- rnorm(n)
  x2 <- sqrt(0.5) * x1 + sqrt(0.5) * rnorm(n)

  res[r, ] <- c(t_stat(y, x1), t_stat(y, x2))
}

print(mean(apply(abs(res), 1, max) > qnorm(0.95)))
```

```
## [1] 0.1648
```

Conventionally, applied economists do not appreciate the problem of variable selection, even though they always select variables implicitly. They rely on their prior knowledge to choose variables from a large number of potential candidates. Recently years economists wake up from the long lasting negligence. Stock and Watson (2012) are concerning about forecasting 143 US macroeconomic indicators. They conduct a horse race of several variable selection methods.

The most well-known variable selection method in regression context is the least-absolute-shrinkage-and-selection-operator (Lasso) (Tibshirani 1996). Upon the usual OLS criterion function, Lasso penalizes the L_1 norm of the coefficients. The criterion function of Lasso is written as

$$(2n)^{-1}(Y - X\beta)'(Y - X\beta) + \lambda\|\beta\|_1$$

where $\lambda \geq 0$ is a tuning parameter. Unlike OLS or ridge regression, Lasso does not have a closed-form solution. Fortunately, it is an convex optimization so numerical optimization is fast and reliable for high-dimensional parameter.

In a wide range of values of λ, Lasso can shrink some coefficients exactly to 0, which suggests that these variables are likely to be irrelevant in the regression. This phenomenon is similar to "corner solution" that we solve utility maximization in microeconomics.

In terms of theoretical property, Zou (2006) finds that Lasso cannot consistently distinguish the relevant variables from the irrelevant ones.

Another successful variable selection method is smoothly-clipped-absolute-deviation (SCAD) (Fan and Li 2001). Its criterion function is

$$(2n)^{-1}(Y - X\beta)'(Y - X\beta) + \sum_{j=1}^{d} \rho_\lambda(|\beta_j|)$$

where

$$\rho'_\lambda(\theta) = \lambda\left\{1\{\theta \leq \lambda\} + \frac{(a\lambda - \theta)_+}{(a-1)\lambda} \cdot 1\{\theta > \lambda\}\right\}$$

for some $a > 2$ and $\theta > 0$. This is a non-convex function, and Fan and Li (2001) establish the so-called *oracle property*. An estimator boasting the oracle property can achieve variable selection consistency and (pointwise) asymptotic normality simultaneously.

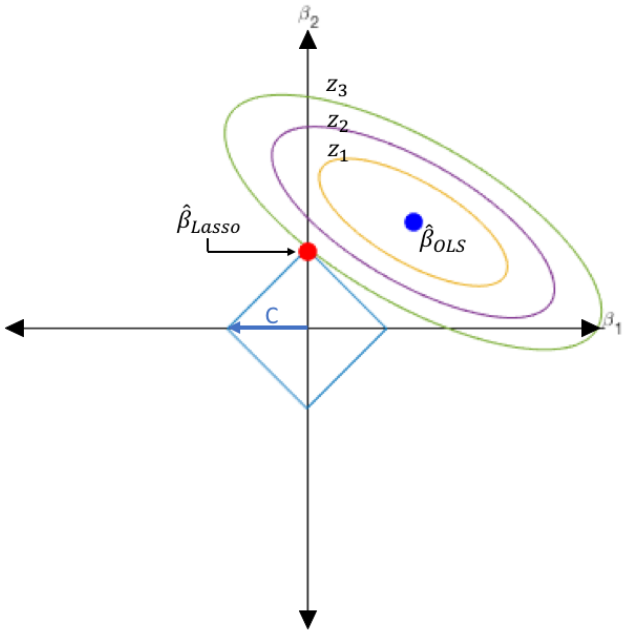


图 6: Lasso with two parameters

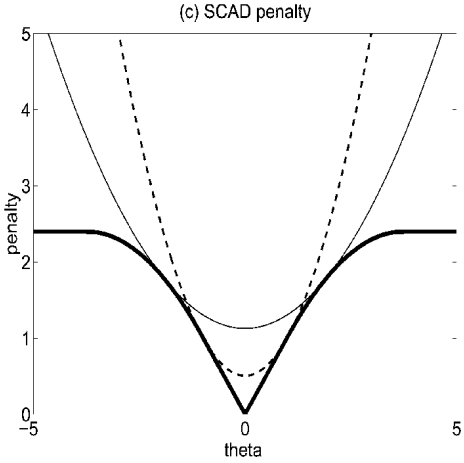


图 7: SCAD criterion and quadratic approximation

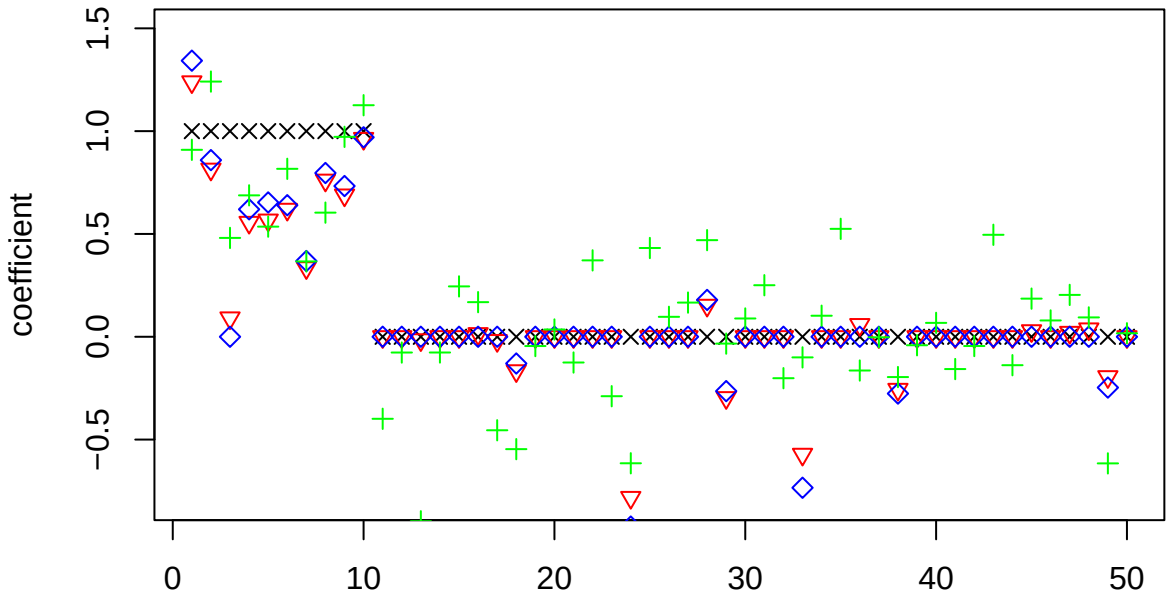
The follow-up *adaptive Lasso* (Zou 2006) also enjoys the oracle property. Adaptive Lasso is a two step scheme: 1. First run a Lasso or ridge regression and save the estimator $\hat{\beta}^{(1)}$. 2. Solve

$$(2n)^{-1}(Y - X\beta)'(Y - X\beta) + \lambda \sum_{j=1}^{d} w_j |\beta_j|$$

where $w_j = 1 \Big/ \left|\hat{\beta}_j^{(1)}\right|^a$ and $a \geq 1$ is a constant. (Common choice is $a = 1$ or 2).

In R, `glmnet` or `LARS` implements Lasso, and `ncvreg` carries out SCAD. Adaptive Lasso can be done by setting the weight via the argument `penalty.factor` in `glmnet`.

```
n <- 40
p <- 50
b0 <- c(rep(1, 10), rep(0, p - 10))
x <- matrix(rnorm(n * p), n, p)
y <- x %*% b0 + rnorm(n)

ols <- MASS::ginv(t(x) %*% x) %*% (t(x) %*% y) # OLS
# Implement Lasso by glmnet
cv_lasso <- glmnet::cv.glmnet(x, y)
lasso_result <- glmnet::glmnet(x, y, lambda = cv_lasso$lambda.min)

# Get weights
b_temp <- as.numeric(lasso_result$beta)
b_temp[b_temp == 0] <- 1e-8
w <- 1 / abs(b_temp) # Let gamma = 1

# Implement Adaptive Lasso by glmnet
cv_alasso <- glmnet::cv.glmnet(x, y, penalty.factor = w)
alasso_result <-
  glmnet::glmnet(x, y, penalty.factor = w, lambda = cv_alasso$lambda.min)

plot(b0, ylim = c(-0.8, 1.5), pch = 4, xlab = "", ylab = "coefficient")
points(lasso_result$beta, col = "red", pch = 6)
points(alasso_result$beta, col = "blue", pch = 5)
points(ols, col = "green", pch = 3)
```




```
# out of sample prediction
x_new <- matrix(rnorm(n * p), n, p)
y_new <- x_new %*% b0 + rnorm(n)
lasso_msfe <- (y_new - predict(lasso_result, newx = x_new)) %>% var()
alasso_msfe <- (y_new - predict(alasso_result, newx = x_new)) %>% var()
ols_msfe <- (y_new - x_new %*% ols) %>% var()

print(c(lasso_msfe, alasso_msfe, ols_msfe))
```

```
## [1] 3.396891 3.823844 6.536441
```

We can DIY Lasso by `CVXR`.

```
library(CVXR)

lambda <- 2 * cv_lasso$lambda.min # tuning parameter

# CVXR for Lasso
beta_cvxr <- Variable(p)
obj <- sum_squares(y - x %*% beta_cvxr) / (2 * n) + lambda * p_norm(beta_cvxr, 1)
prob <- Problem(Minimize(obj))
lasso_cvxr <- solve(prob)
beta_cvxr_hat <- lasso_cvxr$getValue(beta_cvxr) %>% as.vector() %>% print()
```

```
##  [1]  1.099223e+00  6.869818e-01 -1.976718e-22  5.494398e-01  3.374444e-01
##  [6]  5.540718e-01  1.495345e-01  7.616507e-01  5.927460e-01  8.891177e-01
## [11]  2.057997e-23 -3.679063e-22  1.101902e-22  3.448655e-23 -4.299999e-23
## [16]  7.607052e-02  2.628120e-22 -1.997825e-01 -5.715073e-22 -3.502413e-22
## [21] -5.584471e-22  2.555965e-23 -2.462974e-23 -6.034640e-01  1.669944e-22
## [26]  6.661819e-23 -1.160165e-22 -2.095927e-22 -3.007470e-01 -3.447398e-02
## [31]  1.322415e-03 -4.797364e-22 -3.512490e-01 -2.119319e-22  8.883438e-22
## [36]  7.450615e-23  8.032541e-23 -2.690041e-01 -1.349636e-22 -6.714044e-23
## [41]  3.531985e-22 -3.152968e-22  2.093344e-22  9.981627e-23  1.068263e-22
## [46]  2.968204e-22  2.645526e-22  4.984653e-22 -8.375600e-02  6.356453e-23
```

More methods are available if prediction of the response variables is the sole purpose of the regression. An intuitive one is called *stagewise forward selection*. We start from an empty model. Given many candidate x_j, in each round we add the regressor that can produce the biggest R^2. This method is similar to the idea of L_2 componentwise boosting, which does not adjust the coefficients fitted earlier.

6.4 Shrinkage Estimation in Econometrics

- Su, Shi, and Phillips (2016): use shrinkage estimation for classification
- Shi (2016b): convergence rate of GMM Lasso
- Lee, Shi, and Gao (2022): Lasso and adaptive Lasso in predictive regression
- Shi and Huang (2019): forward selection
- Shi, Su, and Xie (2020): latent group in forecast combination

6.5 Empirical Applications

- Lehrer and Xie (2017): movie box office
- Feng, Giglio, and Xiu (2020): factor zoo, compare machine learning methods
- (**chinco2017sparse?**): financial market, Lasso prediction

6.6 Reading

- Efron and Hastie: Ch. 16
- Athey (2018)

6.7 Appendix

Suppose $y_i = x_i'\beta_0 + e_i$, where e_i is independent of x_i and $\text{var}[e_i] = \sigma^2$. Then

$$\min_\beta E[(y_i - x_i'\beta)^2] = E[(y_i - x_i'\beta_0)^2] = E[e_i^2] = \sigma^2.$$

This is the minimal error that can be achieved in the population.

In reality, we have a sample (y_i, x_i) of n observations, and we estimate β by the OLS estimator $\hat{\beta} = (X'X)^{-1}X'y$. The expectation of the SSR is

$$E\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i'\hat{\beta})^2\right] = \frac{1}{n}E\left[e'(I_n - X(X'X)^{-1}X)e\right] = \frac{\sigma^2}{n}(n-p) = \sigma^2\left(1 - \frac{p}{n}\right) < \sigma^2$$

Asymptotically, if $p/n \to 0$, the two risks converge. Otherwise if $p/n \to c$, the expected SSR is strictly smaller than the minimal population risk. The model is overfitted.

6.8 References

7 Prediction-Oriented Algorithms

In this lecture, we introduce supervised learning methods that induces data-driven interaction of the covariates. The interaction makes the covariates much more flexible to capture the subtle feature in the data. However, insufficient theoretical understanding is shed little light on these methods due to the complex nature, so they are often viewed by theorists as "black-boxes" methods. In real applications, when the machines are carefully tuned, they can achieve surprisingly superior performance. Gu, Kelly, and Xiu (2020) showcase a horse race of a myriad of methods, and the general message is that interaction helps with forecast in the financial market. In the meantime, industry insiders are pondering whether these methods are "alchemy" which fall short of scientific standard. Caution must be exercised when we apply these methods in empirical economic analysis.

7.1 Regression Tree and Bagging

We consider supervised learning setting in which we use x to predict y. It can be done by traditional nonparametric methods such as kernel regression. *Regression tree* (Breiman et al. 1984) is an alternative to kernel regression. Regression tree recursively partitions the the space of the regressors. The algorithm is easy to describe: each time a covariate is split into two dummies, and the splitting criterion is aggressive reduction of the SSR. In the formulate of the SSR, the fitted value is computed as the average of y_i's in a partition.

The tuning parameter is the depth of the tree, which is referred to the number of splits. Given a dataset d and the depth of the tree, the fitted regression tree $\hat{r}(d)$ is completely determined by the data.

The problem of the regression tree is its instability. For data generated from the same data generating process (DGP), the covariate chosen to be split and the splitting point can vary widely and they heavily influence the path of the partition.

Bootstrap averaging, or *bagging* for short, was introduced to reduce the variance of the regression tree (Breiman 1996). Bagging grows a regression tree for each bootstrap sample, and then do a simple average. Let $d^{\star b}$ be the b-th bootstrap sample of the original data d, and then the bagging estimator is defined as

$$\hat{r}_{\text{bagging}} = B^{-1}\sum_{b=1}^{B}\hat{r}(d^{\star b}).$$

Bagging is an example of the *ensemble learning*.

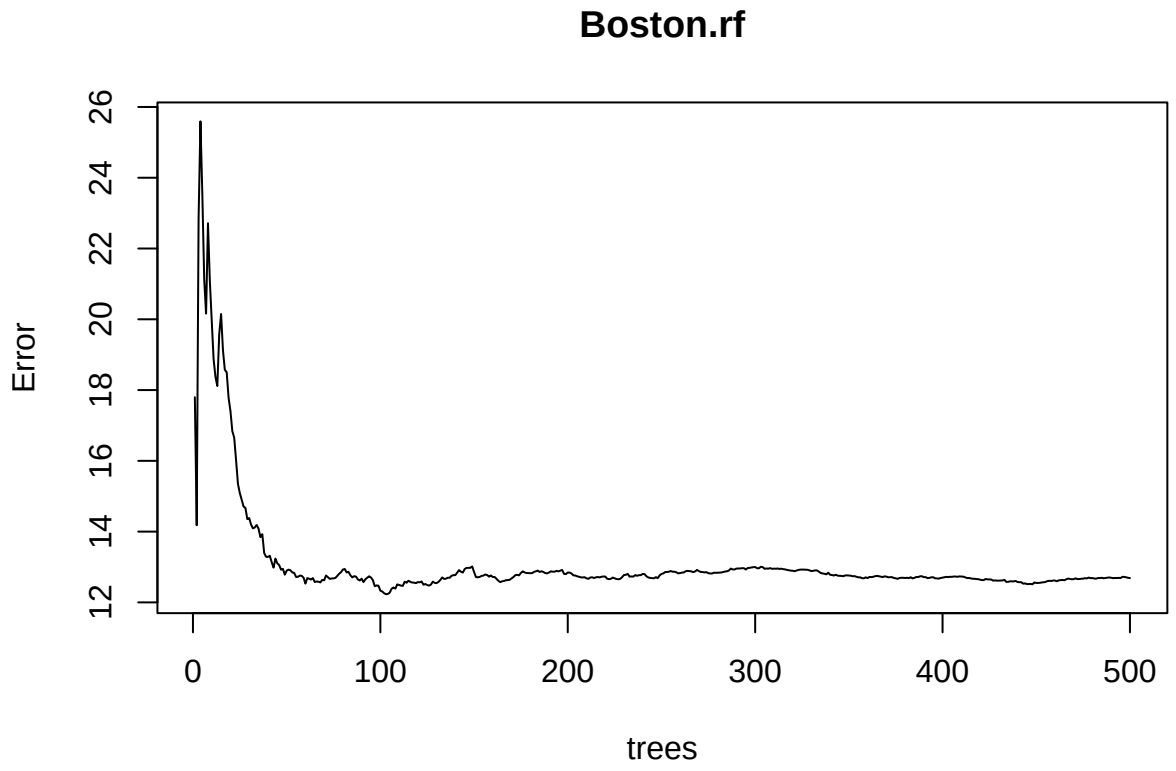
Inoue and Kilian (2008) is an early application of bagging in time series forecast. Hirano and Wright (2017) provide a theoretical perspective on the risk reduction of bagging.

7.2 Random Forest

Random forest (Breiman 2001a) shakes up the regressors by randomly sampling m out of the total p covarites before *each split of a tree*. The tuning parameters in random forest is the tree depth and m. Due to the "de-correlation" in sampling the regressors, it in general performs better than bagging in prediction exercises.

Below is a very simple real data example of random forest using the Boston Housing data.

```
require(randomForest)
require(MASS) # Package which contains the Boston housing dataset
attach(Boston)
set.seed(101)

# training Sample with 300 observations
train <- sample(1:nrow(Boston), 300)

Boston.rf <- randomForest(medv ~ ., data = Boston, subset = train)
plot(Boston.rf)
```




```
importance(Boston.rf)
```

```
##         IncNodePurity
## crim       1373.07707
## zn          247.35123
## indus      1326.18996
## chas         47.87453
## nox        1527.38467
## rm         6075.29521
## age        1031.69446
## dis        1551.31102
## rad         164.36409
## tax         788.77341
## ptratio    1345.75212
## black       526.37109
## lstat      6214.98744
```

Despite the simplicity of the algorithm, the consistency of random forest is not proved until Scornet, Biau, and Vert (2015), and inferential theory was first established by Wager and Athey (2018) in the context of treatment effect estimation. Athey, Tibshirani, and Wager (2019) generalizes CART to local maximum likelihood.

Example: Random forest for Survey of Professional Forecasters in `data_example/SPF_RF.R` from Cheng, Huang, and Shi (2020). The script uses `caret` framework.

7.3 Gradient Boosting

Bagging and random forest almost always use equal weight on each generated tree for the ensemble. Instead, *tree boosting* takes a distinctive scheme to determine the ensemble weights. It is a deterministic approach that does not resample the original data.

1. Use the original data $d^0 = (x_i, y_i)$ to grow a shallow tree $\hat{r}^0(d^0)$. Save the prediction $f_i^0 = \alpha \cdot \hat{r}^0(d^0, x_i)$ where $\alpha \in [0, 1]$ is a shrinkage tuning parameter. Save the residual $e_i^0 = y_i - f_i^0$. Set $m = 1$.
2. In the m-th iteration, use the data $d^m = (x_i, e_i^{m-1})$ to grow a shallow tree $\hat{r}^m(d^m)$. Save the prediction $f_i^m = f_i^{m-1} + \alpha \cdot \hat{r}^m(d, x_i)$. Save the residual $e_i^m = y_i - f_i^m$. Update $m = m + 1$.
3. Repeat Step 2 until $m > M$.

In this boosting algorithm there are three tuning parameters: the tree depth, the shrinkage level α, and the number of iterations M. The algorithm can be sensitive to all the three tuning parameters. When a model is tuned well, it often performs remarkably. For example, the script `Beijing_housing_gbm.R` achieves much higher out-of-sample R^2 than OLS, reported in Lin et al. (2020). This script implements boosting via the package `gbm`, which stands for "Gradient Boosting Machine."

There are many variants of boosting algorithms. For example, L_2-boosting, componentwise boosting, and AdaBoosting, etc. Statisticians view boosting as a gradient descent algorithm to reduce the risk. The fitted tree in each iteration is the deepest descent direction, while the shrinkage tames the fitting to avoid proceeding too aggressively.

- Shi (2016a) proposes a greedy algorithm in similar spirit to boosting for moment selection in GMM.
- Phillips and Shi (2021) uses L_2-boosting as a boosted version of the Hodrick-Prescott filter.
- Shi and Huang (2019)

7.4 Neural Network

A neural network is the workhorse behind Alpha-Go and self-driven cars. However, from a statistician's point of view it is just a particular type of nonlinear models. Figure 1 illustrates a one-layer neural network, but in general there can be several layers. The transition from layer $k-1$ to layer k can be written as

$$\begin{aligned} z_l^{(k)} &= w_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} w_{lj}^{(k-1)} a_j^{(k-1)} \\ a_l^{(k)} &= g^{(k)}(z_l^{(k)}), \end{aligned} \tag{7}$$

where $a_j^{(0)} = x_j$ is the input, $z_l^{(k)}$ is the k-th hidden layer, and all the ws are coefficients to be estimated. The above formulation shows that $z_l^{(k)}$ usually takes a linear form, while the *activation function* $g(\cdot)$ can be an identity function or a simple nonlinear function. Popular choices of the activation function are sigmoid $(1/(1+\exp(-x)))$ and rectified linear unit (ReLu, $z \cdot 1\{x \geq 0\}$), etc.

A user has several decisions to make when fitting a neural network: besides the activation function,

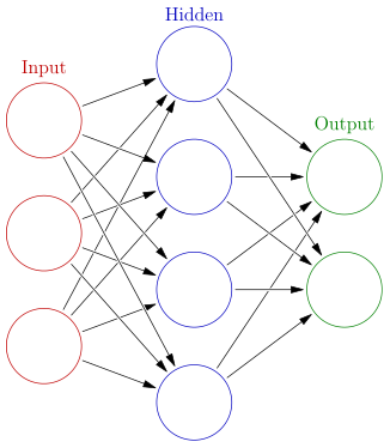


图 8: A Single Layer Feedforward Neural Network (from Wiki)

the tuning parameters are the number of hidden layers and the number of nodes in each layer. Many free parameters are generated from the multiple layer and multiple nodes, and in estimation often regularization methods are employed to penalize the l_1 and/or l_2 norms, which requires extra tuning parameters. `data_example/Keras_ANN.R` gives an example of a neural network with two hidden layers, each has 64 nodes, and the ReLu activation function.

Due to the nonlinear nature of the neural networks, theoretical understanding about its behavior is still scant. One of the early contributions came from econometrician: Hornik, Stinchcombe, and White (1989) (Theorem 2.2) show that a single hidden layer neural network, given enough many nodes, is a *universal approximator* for any measurable function.

After setting up a neural network, the free parameters must be determined by numerical optimization. The nonlinear complex structure makes the optimization very challenging and the global optimizer is beyond guarantee. In particular, when the sample size is big, the de facto optimization algorithm is the stochastic gradient descent.

Thanks to computational scientists, Google's `tensorflow` is a popular backend of neural network estimation, and `keras` is the deep learning modeling language. Their relationship is similar to `Mosek` and `CVXR`.

7.5 Stochastic Gradient Descent

In optimization we update the D-dimensional parameter

$$\beta_{k+1} = \beta_k + a_k p_k,$$

where $a_k \in \mathbb{R}$ is the step length and $p_k \in \mathbb{R}^D$ is a vector of directions. Use a Talyor expansion,

$$f(\beta_{k+1}) = f(\beta_k + a_k p_k) \approx f(\beta_k) + a_k \nabla f(\beta_k) p_k,$$

If in each step we want the value of the criterion function $f(x)$ to decrease, we need $\nabla f(\beta_k) p_k \leq 0$. A simple choice is $p_k = -\nabla f(\beta_k)$, which is called the deepest decent. Newton's method corresponds to $p_k = -(\nabla^2 f(\beta_k))^{-1} \nabla f(\beta_k)$, and BFGS uses a low-rank matrix to approximate $\nabla^2 f(\beta_k)$. The

linear search is a one-dimensional problem and it can be handled by either exact minimization or backtracking. Details of the descent method is referred to Chapter 9.2–9.5 of Boyd and Vandenberghe (2004).

When the sample size is huge and the number of parameters is also big, the evaluation of the gradient can be prohibitively expensive. Stochastic gradient descent (SGD) uses a small batch of the sample to evaluate the gradient in each iteration. It can significantly save computational time. It is the *de facto* optimization procedure in complex optimization problems such as training a neural network.

However, SGD involves tuning parameters (say, the batch size and the learning rate. Learning rate replaces the step length a_k and becomes a regularization parameter.) that can dramatically affect the outcome, in particular in nonlinear problems. Careful experiments must be carried out before serious implementation.

Below is an example of SGD in the PPMLE that we visited in the lecture of optimization, now with sample size 100,000 and the number of parameters 100. SGD is usually much faster than `nlopt`.

The new functions are defined with the data explicitly as arguments. Because in SGD each time the log-likelihood function and the gradient are evaluated at a different subsample.

```
poisson.loglik <- function(b, y, X) {
  b <- as.matrix(b)
  lambda <- exp(X %*% b)
  ell <- -mean(-lambda + y * log(lambda))
  return(ell)
}


poisson.loglik.grad <- function(b, y, X) {
  b <- as.matrix(b)
  lambda <- as.vector(exp(X %*% b))
  ell <- -colMeans(-lambda * X + y * X)
  ell_eta <- ell
  return(ell_eta)
}
```

```
##### generate the artificial data
set.seed(898)
nn <- 1e5
K <- 100
X <- cbind(1, matrix(runif(nn * (K - 1)), ncol = K - 1))
b0 <- rep(1, K) / K
y <- rpois(nn, exp(X %*% b0))

b.init <- runif(K)
b.init <- 2 * b.init / sum(b.init)
# and these tuning parameters are related to N and K

n <- length(y)
test_ind <- sample(1:n, round(0.2 * n))
```

```r
y_test <- y[test_ind]
X_test <- X[test_ind, ]

y_train <- y[-test_ind ]
X_train <- X[-test_ind, ]
```

```r
# optimization parameters
# sgd depends on

# * eta: the learning rate
# * epoch: the averaging small batch
# * the initial value

max_iter <- 5000
min_iter <- 20
eta <- 0.01
epoch <- round(100 * sqrt(K))

b_old <- b.init

pts0 <- Sys.time()
# the iteration of gradient
for (i in 1:max_iter) {
  loglik_old <- poisson.loglik(b_old, y_train, X_train)
  i_sample <- sample(1:length(y_train), epoch, replace = TRUE)
  b_new <- b_old - eta * poisson.loglik.grad(b_old, y_train[i_sample], X_train[i_sample, ])
  loglik_new <- poisson.loglik(b_new, y_test, X_test)
  b_old <- b_new # update

  criterion <- loglik_old - loglik_new

  if (criterion < 0.0001 & i >= min_iter) break
}
cat("point estimate =", b_new, ", log_lik = ", loglik_new, "\n")
```

```
## point estimate = 0.007256433 0.003703304 -0.001034752 0.01448194 -0.002049049 0.004100504 -0
```

```r
pts1 <- Sys.time() - pts0
print(pts1)
```

```
## Time difference of 3.150093 mins
```

```r
# optimx is too slow for this dataset.
# Nelder-Mead method is too slow for this dataset

# thus we only sgd with NLoptr

opts <- list("algorithm" = "NLOPT_LD_SLSQP", "xtol_rel" = 1.0e-7, maxeval = 5000)
```

```
pts0 <- Sys.time()
res_BFGS <- nloptr::nloptr(
  x0 = b.init,
  eval_f = poisson.loglik,
  eval_grad_f = poisson.loglik.grad,
  opts = opts,
  y = y_train, X = X_train
)
pts1 <- Sys.time() - pts0
print(pts1)
```

```
## Time difference of 7.17868 secs
```

```
b_hat_nlopt <- res_BFGS$solution

#### evaluation in the test sample
cat("log lik in test data by sgd = ", poisson.loglik(b_new, y = y_test, X_test), "\n")
```

```
## log lik in test data by sgd =  0.8159186
```

```
cat("log lik in test data by nlopt = ", poisson.loglik(b_hat_nlopt, y = y_test, X_test), "\n")
```

```
## log lik in test data by nlopt =  0.8158423
```

```
cat("log lik in test data by true para. = ", poisson.loglik(b0, y = y_test, X_test), "\n")
```

```
## log lik in test data by true para. =  0.8151091
```

7.6 Reading

- Efron and Hastie: Chapter 8, 17 and 18.

7.7 Quotation

7.8 References

8 Data Processing

It is a long work flow from collecting raw data to presenting the statistical evidence of the final empirical results. The process is very much like cooking. (The key difference between cooking and empirical research the level of innovation. In daily food cooking no one blames you if you strictly follows a recipe, but, except replication works, research demands at least some level of originality.)

1. We come up with an idea what dish we would like to present.
2. We go to a supermarket to buy the materials such as meat and vegetables.

3. We clean and prepare the ingredient.
4. We heat the pot and fry the materials until they are edible.
5. We arrange the cooked food into a plate, decorate a little bit if needed, and present it onto a dining table.

Step 4 is the highlight as it turns the materials into editable food. However, it only takes a small fraction of time in the whole steps of food cooking. Most of the time is indeed spend in shopping the raw material and prepare. In this chapter, we talk about the underappreciated but extremely important step.

I would like to emphasize the replicability of empirical research. Empirical research is never a linear process: we need to constantly come back to revisit the prior steps. We need to guarantee that the empirical results replicable. This is a requirement for any scientific finding. Only if the results are replicable by the authors themselves and other researchers we can start a meaningful discussion of the research design and the credibility of the empirical findings. Unreplicable research is voodoo.

Data processing is an art instead of science. Just like writing, everyone can has his or her own style. However, some styles are inefficient and should be avoided. Here are some practical guidance that I find useful in my research, and I would like to share with you.

8.1 Data Collection

If the data is collected somewhere online, keep a log so that you know where it is downloaded so that we can easily repeat the process if necessary.

Keep the raw data without any editing. If editing is essential, save the edited data into another file. Manual editing should be avoided whenever possible; it is always recommended editing being done by a script. This is why code-driven software like R or STATA is preferred over menu-driven statistical software like Excel or SPSS.

If the data processing takes many steps and it takes time to run, you can save intermediate datasets as some milestones.

Write comments about what a script does. Keep a separate log.

8.2 References

9 Git

Git is a version control system useful when developing and maintaining coding projects as well as preparing long documents.

Git helps me manage course materials, and I use Github https://github.com/zhentaoshi/econ5170 to share them. Free Git books and tutorials are available online. Self-learning is important.

- Atlassian Online Tutorial
- Udacity Course
- CodeAcademy Course

- Pro Git
- Many video tutorials, for example this one.

We introduce some essential Git commands. There are two ways to interact with Git. Git provides a command line tool Git Bash, and there are many free Git GUIs available (I recommend SourceTree). Even if we use a GUI, knowing the basic commands is helpful.

9.1 Basic Commands

9.1.1 Local

- `git status` inspects the contents of the working directory and staging area.
- `git add filename` adds files to the staging area from working directory.
- `git add filename1 filename2` adds multiple files to the staging area.
- If the file is changed, we can use `git diff filename` to see the difference between the file in the working directory and the staging area. Reuse `git add filename` to add the updated file to the staging area.
- `git commit` stores changes from the staging area to the repository.
- `git commit -m "Commit Message"` The commit message must be in the quotation marks.
- `git log` displays historical commits stored chronologically in the repository.
- `git config --global user.name <name>`
- `git config --global user.email <email>`
- `git tag -a v1.0 -m 'message' [optional:commit-id]`
- `git rm --cached filename`
- `git branch [brach_name]`
- `git checkout [commit_id or branch name]`
- .gitignore

9.1.1.1 Eraser-like features

The latest commit is called **HEAD** commit * `git show HEAD` to view the HEAD commit. * `git checkout HEAD filename` to restore the file in the working directory to what you made in last commit. * `git reset HEAD filename` this command unstages the file from committing in the staging area. This command resets the file in the staging area to be the same as the HEAD commit. It does not discard file changes from the working directory, it just removes them from the staging area. * `git reset SHA` This command works by using **the first 7 characters** of the SHA of a previous commit.

9.1.1.2 Branch

- `git branch` displays the current branch.
- `git branch brach_name` creates a new branch. The branch name cannot contain white-space
- `git checkout branch_name` switch to a branch
- `git merge branch_name` merging the branch into master **merge conflict**: Git doesn't know which changes you want to keep. Modify the file in the working directory and commit it again to avoid conflict.
- `git branch -d branch_name` deletes a branch from the project.

In Git, branches are usually a means to an end. We create them to work on a new project feature, but the ultimate goal is to merge that feature into the master branch.

9.1.2 Remote

- `git clone https://github.com/zhentaoshi/econ5170`
- `git remote add origin` adds the origin remote's URL.
- `git remote -v` lists git project's remote copies.
- `git push origin master` upload local commits to the remote repository.
- `git pull` download the remote copy and merge.
- `git fetch` fetches the remote copy to the local hard disk. **Note:** This command will not merge changes from the remote into your local repository. It inspects the changes on the remote branch.
- `git remote set-url origin https://github.com/zhentaoshi/econ5170` changes the remote's URL.

Collaboration typically works as follows: 1. Fetch and merge changes from the remote; 2. Create a branch to work on a new project feature; 3. Develop the feature on your branch and commit your work; 4. Fetch and merge from the remote again (in case new commits have been uploaded while you were working); 5. Push your branch up to the remote for review.

9.1.2.1 Conflict Collaborators working on the same file separately might cause conflicts. In such situations, the command `git pull` will not merge chanegs from the remote into your local repository automatically due to the conflict. You are expected to see a similar message on the shell:

```
git pull
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://bitbucket.org/your-user-name/repo-name
   9b02654..0462f26  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

If you check the status by `git status`, you would be told that you have unmerged paths.

```
git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
```

```
  (use "git add <file>..." to mark resolution)

        both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

To resolve the conflicts, we simply use a text editor, say *Sublime Text 3*, *Atom* or *Visual Studio Code*, to open the file with conflicts. In the illustrative example, it is the `README.md` file.

```
# Conflict-Resolving

<<<<<<< HEAD
Local changes.
=======
Remote changes.
>>>>>>> 0462f26ddfe83c35424168c2d7a0bed62c653413
```

You can see conflicts indicated by Git. The content between `<<<<<<< HEAD` and `=======` is on HEAD and content between `=======` and `>>>>>>>` is from the commit `0462f26ddfe83c35424168c2d7a0bed62c653413`. To resolve the conflicts, you just remove `<<<<<<< HEAD`, `=======` and `>>>>>> 0462f26ddfe83c35424168c2d7a0bed6` and keep what you want keep in the file.

In some cases, the difference between the local branch and the remote branch is significant, or the file with conflicts is of a special format, say Lyx file. It might be cumbersome to resolve all conflicts in the text editor. Another way to resovle conflicts is simply to open two versions of the files and copy and paste new updates and then commit the updated file. It easy to do so in *SourceTree*: select the commit -> right click the file -> click `Open Selected Version`.

In addition, we can resolve conflicts with *SourceTree* (**preferred**) or external merge tools, for example *KDiff3*. For details, it is recommended to refer to the answer on Stack Overflow.

Acknowledgment: Part of this notes is based on the course offered by CodeAcademy.

References

Athey, Susan. 2018. "The Impact of Machine Learning on Economics." In *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press.

Athey, Susan, Julie Tibshirani, and Stefan Wager. 2019. "Generalized Random Forests." *The Annals of Statistics* 47 (2): 1148–78.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge university press.

Breiman, Leo. 1996. "Bagging Predictors." *Machine Learning* 24 (2): 123–40.

———. 2001a. "Random Forests." *Machine Learning* 45 (1): 5–32.

———. 2001b. "Statistical Modeling: The Two Cultures." *Statistical Science* 16 (3): 199–231.

Breiman, Leo, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 1984. *Classification and Regression Trees*. Wadsworth Publishing.

Cameron, A Colin, and Pravin K Trivedi. 2005. *Microeconometrics: Methods and Applications*. Cambridge University Press.

Chang, Yoosoon. 2004. "Bootstrap Unit Root Tests in Panels with Cross-Sectional Dependency." *Journal of Econometrics* 120 (2): 263–93.

Cheng, Kayan, Naijing Huang, and Zhentao Shi. 2020. "Survay-Based Forecasting: To Average or Not to Average." In *Studies in Computational Intelligence: Behavioral Predictive Modeling in Econometrics*, edited by Woraphon Yamaka Vladik Kreinovich Songsak Sriboonchitta. Springer-Verlag.

Chernozhukov, Victor, and Han Hong. 2003. "An MCMC Approach to Classical Estimation." *Journal of Econometrics* 115 (2): 293–346.

Davidson, Russell, and James G MacKinnon. 2010. "Wild Bootstrap Tests for IV Regression." *Journal of Business & Economic Statistics* 28 (1): 128–44.

Efron, B. 1979. "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics* 7 (1): 1–26.

Fan, Jianqing, and Runze Li. 2001. "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties." *Journal of the American Statistical Association* 96 (456): 1348–60.

Feng, Guanhao, Stefano Giglio, and Dacheng Xiu. 2020. "Taming the Factor Zoo: A Test of New Factors." *The Journal of Finance* 75 (3): 1327–70.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2008. *The Elements of Statistical Learning*. 2nd ed. Springer.

Fu, Anqi, Balasubramanian Narasimhan, and Stephen Boyd. 2019. "CVXR: An r Package for Disciplined Convex Optimization." *Journal of Statistical Software*.

Gourieroux, Christian, Alain Monfort, and Eric Renault. 1993. "Indirect Inference." *Journal of Applied Econometrics* 8 (S1): S85–118.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. 2020. "Empirical Asset Pricing via Machine Learning." *The Review of Financial Studies* 33 (5): 2223–73.

Guo, Naijia, Xiaoyu Xia, and Junsen Zhang. 2018. "A Matching Model of Intergenerational Co-Residence and Its Application in China." The Chinese University of Hong Kong.

Hall, Peter, and Joel L Horowitz. 1996. "Bootstrap Critical Values for Tests Based on Generalized-Method-of-Moments Estimators." *Econometrica*, 891–916.

Heckman, James J. 1977. "Sample Selection Bias as a Specification Error (with an Application to the Estimation of Labor Supply Functions)." National Bureau of Economic Research Cambridge, Mass., USA.

Hirano, Keisuke, and Jonathan H Wright. 2017. "Forecasting with Model Uncertainty: Representations and Risk Reduction." *Econometrica* 85 (2): 617–43.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–66.

Inoue, Atsushi, and Lutz Kilian. 2008. "How Useful Is Bagging in Forecasting Economic Time Series? A Case Study of US Consumer Price Inflation." *Journal of the American Statistical Association* 103 (482): 511–22.

Judd, Kenneth L. 1998. *Numerical Methods in Economics*. MIT press.

Lee, Ji Hyung, Zhentao Shi, and Zhan Gao. 2022. "On LASSO for Predictive Regression." 2. *Journal of Econometrics*. Vol. 229. Elsevier.

Lehrer, Steven, and Tian Xie. 2017. "Box Office Buzz: Does Social Media Data Steal the Show from Model Uncertainty When Forecasting for Hollywood?" *Review of Economics and Statistics* 99 (5): 749–55.

Li, Quefeng, Guang Cheng, Jianqing Fan, and Yuyan Wang. 2018. "Embracing the Blessing of Dimensionality in Factor Models." *Journal of the American Statistical Association* 113 (521): 380–89.

Li, Tong. 2010. "Indirect Inference in Structural Econometric Models." *Journal of Econometrics* 157 (1): 120–28.

Lin, Wei, Zhentao Shi, Yishu Wang, and Ting Hin Yan. 2020. "Unfolding Beijing in a Hedonic

Way." The Chinese University of Hong Kong. https://www.researchgate.net/publication/33955 1353_Unfolding_Beijing_in_a_Hedonic_Way.

Mullainathan, Sendhil, and Jann Spiess. 2017. "Machine Learning: An Applied Econometric Approach." *Journal of Economic Perspectives* 31 (2): 87–106.

Nash, John C. 2014. "On Best Practice Optimization Methods in r." *Journal of Statistical Software* 60 (2): 1–14.

Owen, Art B. 1988. "Empirical Likelihood Ratio Confidence Intervals for a Single Functional." *Biometrika* 75 (2): 237–49.

Pakes, Ariel, and David Pollard. 1989. "Simulation and the Asymptotics of Optimization Estimators." *Econometrica*, 1027–57.

Phillips, Peter CB. 2012. "Folklore Theorems, Implicit Maps, and Indirect Inference." *Econometrica* 80 (1): 425–54.

Phillips, Peter CB, and Zhentao Shi. 2021. "Boosting: Why You Can Use the HP Filter." *International Economic Review* 62 (2): 521–70.

Qin, Jin, and Jerry Lawless. 1994. "Empirical Likelihood and General Estimating Equations." *The Annals of Statistics* 22 (1): 300–325.

Roy, Andrew Donald. 1951. "Some Thoughts on the Distribution of Earnings." *Oxford Economic Papers* 3 (2): 135–46.

Scornet, Erwan, G茅rard Biau, and Jean-Philippe Vert. 2015. "Consistency of Random Forests." *The Annals of Statistics* 43 (4): 1716–41.

Shi, Zhentao. 2016a. "Econometric Estimation with High-Dimensional Moment Equalities." *Journal of Econometrics* 195 (1): 104–19.

———. 2016b. "Estimation of Sparse Structural Parameters with Many Endogenous Variables." *Econometric Reviews* 35 (8-10): 1582–1608.

Shi, Zhentao, and Jingyi Huang. 2019. "Forward-Selected Panel Data Approach for Program Evaluation." *arXiv Preprint arXiv:1908.05894*.

Shi, Zhentao, Liangjun Su, and Tian Xie. 2020. "High Dimensional Forecast Combinations Under Latent Structures." *arXiv* 2010.09477.

Shi, Zhentao, and Huanhuan Zheng. 2018. "Structural Estimation of Behavioral Heterogeneity." *Journal of Applied Econometrics* 33 (5): 690–707.

Silva, JMC Santos, and Silvana Tenreyro. 2006. "The Log of Gravity." *The Review of Economics and Statistics* 88 (4): 641–58.

Smith, Anthony A. 1993. "Estimating Nonlinear Time-Series Models Using Simulated Vector Autoregressions." *Journal of Applied Econometrics* 8 (S1): S63–84.

Stock, James H, and Mark W Watson. 2012. "Generalized Shrinkage Methods for Forecasting Using Many Predictors." *Journal of Business & Economic Statistics* 30 (4): 481–93.

Su, Liangjun, Zhentao Shi, and Peter CB Phillips. 2016. "Identifying Latent Structures in Panel Data." *Econometrica* 84 (6): 2215–64.

Taddy, Matt. 2018. "The Technological Elements of Artificial Intelligence." In *The Economics of Artificial Intelligence: An Agenda*, 61–87. University of Chicago Press.

Theu脽l, Stefan, Florian Schwendinger, and Kurt Hornik. 2019. "ROI: The r Optimization Infrastructure Package." Research Report Series / Department of Statistics and Mathematics 133. Vienna: WU Vienna University of Economics; Business. http://epub.wu.ac.at/5858/.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–88.

Wager, Stefan, and Susan Athey. 2018. "Estimation and Inference of Heterogeneous Treatment Effects Using Random Forests." *Journal of the American Statistical Association* 113 (523): 1228–42.

Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc.

Zou, Hui. 2006. "The Adaptive Lasso and Its Oracle Properties." *Journal of the American Statistical Association* 101 (476): 1418–29.