Research Methods in Political Science I

Introduction to Literate Programming with R Markdown

Yuki Yanai

October 17, 2015

Contents

1 How to Use R Markdown **1**

1.1 Markdown 2

1.1.1 Mathematical Formulae 3

1.2 R Code Chunks 3

1.3 Transform R Markdown Files 5

1.3.1 R Makdown to HTML 5

1.3.2 R Markdown to PDF 6

2 Example of R Markdown: Simulations in R **6**

2.1 Variances and Unbiased Variances 6

2.1.1 Setting Up the Simulations 6

2.2 Write Functions to Run Simulations 7

1 How to Use R Markdown

You will learn to use R Markdown here. **Compare the R Markdown file (Literate-Programming.Rmd) to its output html file (Leterate-Programming.html: this page!) or pdf file Literate-Programming.pdf**. You should open the Rmd file with RStudio.

To use this R Markdown file, you have to save the CSS file I wrote (my-markdown.css) in the directory where you have the R Markdown file. To customize the style (i.e. appearance of the html page) of the output, modify the CSS file (see, e.g., this website to learn how to customize CSS). If you would like to use the default style (not recommended), delete the line for CSS option in the header (the 9th line of the .Rmd file).

1.1 Markdown

In Markdown (and R Markdown), you can simply write your sentences as usual. You can make some words italic like *this is italic* or *this is also italic*. You can also use the bold font like **this** or **this**. The bold italic can be used like ***this*** or ***this***.

To begin a new line, insert a line between sentences.

You can create bullet points as follows.

- Item 1
- Item 2
    - Item 2-1
    - Item 2-2

Alternatively,

- Item 1
    - Item 1-1
    - Item 1-2
- Item 2

A single space is necessary after * or -. To make nested lists, indent blocks by tab.

Numbered lists can be created as follows.

1. First item
2. Second item
    1. What?
    2. How?
3. Third item

Note that the numbers you entere only indicate that the list is numbered. The odered numbers are automatically assigned in the output, so you don't have to worry about the numbering. It might be a good practice to use only "1" for numbered lists in order to make re-ordering easy.

You can paste a link like this: Yuki Yanai's Website.

You can insert an image in the page:

(The image file is here)

1.1.1 Mathematical Formulae

You can write math formulae as you do in LaTeX. To write an inline formula, type LaTeX style formula between \$s. E.g., $\bar{x} = \sum_{i=1}^{n} x_i/n$. To write formulae in an independent block, type LaTeX style formulae between \$\$s. E.g.,

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}.$$

1.2 R Code Chunks

In R Markdown files, you write R codes in blocks called *code chunks*. A simple code chunk is like this:

```
a <- 1:10
b <- -1:-10
```

As this example shows, the code chunk starts and ends with three backquotes (“‘) (Note that the end mark must be three backquotes too, not three quotes). After the first three backquotes, write {r} to tell the program that it is a chunk for R codes.After “r” and a space, you should write the name of a chunk. You have to give a unique name to each chunk.

To insert an R code (without the output) in a sentence, write, for instance, “you can obtain the mean of x by mean(x)”. To show the outcome (evaluated value) in a sentence, insert “r” before the command: “the mean of *a* is 5.5”.

You can include figures and tables in R Markdown files.

```
plot(a, b, main = "Scatter␣plot␣of␣b␣vs.␣a")
```

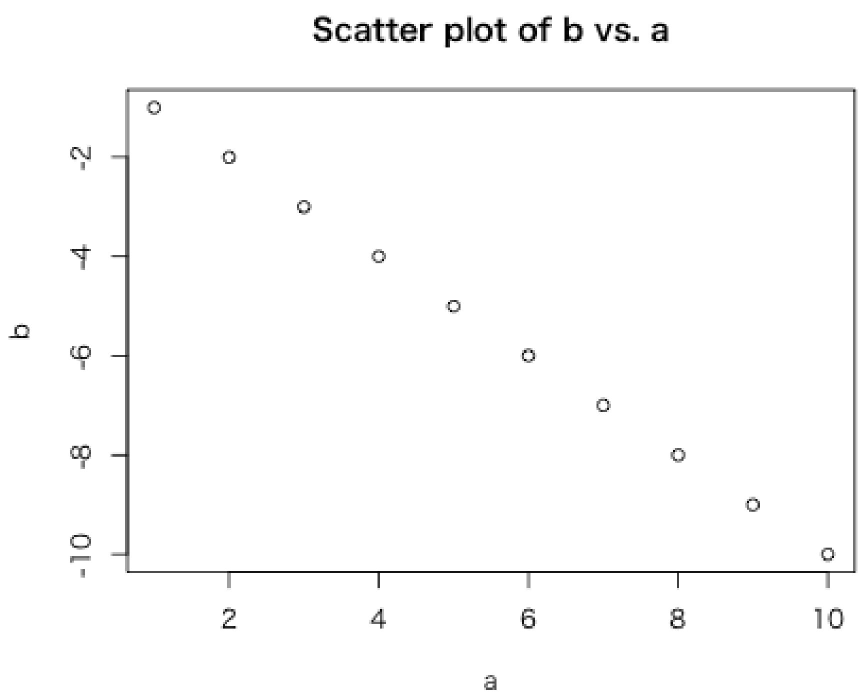

Figure 1: plot of chunk example-plot

By default, R evaluates the chunk at the first line(s), print the outcome of the line(s), and then moves to the next line. E.g.,

```
sd(a)
```

```
## [1] 3.02765
```

```
var(a)
```

```
## [1] 9.166667
```

As you can see, the result of sd(a) is printed before var(a) is evaluated.

To show results of the chunk together, set a chunk option **results** to 'hold'. Chunk options are specified after the chunk names and a comma.

```
sd(a)
var(a)
```

```
## [1] 3.02765
## [1] 9.166667
```

For more information abour R Markdown, visit this website.

1.3 Transform R Markdown Files

1.3.1 R Makdown to HTML

To transform an R Markdown file to an HTML file, you use knitr :: knit2htm() function.

If you have not yet install the **knitr** package, get it from CRAN.

```
install.packages("knitr", dependencies = TRUE)
```

Japanese Windows users might want a development version of knitr to use Japanese characters. Get them by

```
install.packages("devtools", dependencies = TRUE)
library("devtools")
install_github("yihui/knitr")
```

Now, let's convert "Literate-Programming.Rmd" into "Literate-Programming.html".

```
library("knitr")
knit2html("Literate-Programming.Rmd", output = "Literate-Programming.html")
```

Alternatively, if you open an Rmd file in RStduio, you can covert it to an HTML file by clicking the "Knit HTML" button on top. You will find the output HTML file in your working directory. Open it with your web browser!

1.3.2 R Markdown to PDF

You can make a PDF file form an R Markdown file. However, you need **LaTeX** and **Pandoc**. If your computer has them both, you can convert "Literate-Programming.Rmd" to "Literate-Programming.pdf" as follows.

```
## convert Rmd to md by knitr
knit("Literate-Programming.Rmd")
## convert md to PDF by pandoc
system("pandoc -s --toc --number-sections --listings -V documentclass=article --
```

You will find the output PDF file in your working directory. This is the outcome (it needs some modifications to become a good document, though).

Note: To convert an Rmd file containing Japanese characters to a PDF, you need to take care of a few more settings.

2 Example of R Markdown: Simulations in R

2.1 Variances and Unbiased Variances

Suppose the population variance of a random variabe X is σ^2. Let s^2 denote the sample variance of X:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}.$$

Then, the expected value of the sample variance is

$$\mathrm{E}(s^2) = \frac{n-1}{n}\sigma^2.$$

This shows that s^2 is *not* the unbiased estimator of σ^2. The unbiased estimator of σ^2 is u^2:

$$u^2 = \frac{n}{n-1}s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}.$$

It is easy to prove that $\mathrm{E}(u^2) = \sigma^2$, but let's verify it by simulations.

2.1.1 Setting Up the Simulations

First, load the packages we will use.

```
library("ggplot2")
library("dplyr")
```

Then, let's specify the sample size (n), the number of trials in a simulation, and the value of the population variance (σ^2). In addition, let's make a variable to save the simulation results.

```
n <- 10           ## sample size
trials <- 1000    ## number of samples
sigma2 <- 10      ## True Variance
# prepare vectors to save the results
s2 <- rep(NA, trials)  ## vector to save sample variance
u2 <- rep(NA, trials)  ## vector to save unbiased variance
```

Now we are ready. Let's run a simulation. Here, we use **for loop**.

```
for (i in 1:trials) { ## loop for the simulation, i = 1, 2, ..., trials
    x <- rnorm(n, mean = 0, sd = sqrt(sigma2))  ## random sample from N(0, sigma
    s2[i] <- sum((x - mean(x))^2) / n           ## sample variance
    u2[i] <- sum((x - mean(x))^2) / (n - 1)    ## unbiased variance
}
rm(x)  ## remove x since we won't use it
```

Once the simulation is done, let's examine the results.

```
## variance
mean(s2)  ## mean of the sample variannce

## [1] 8.960817

sd(s2)     ## sd of the sample variance

## [1] 4.188909

mean(u2)  ## mean of the unbiased variance

## [1] 9.956464

sd(u2)     ## sd of the unbiased variance

## [1] 4.654343
```

As this example shows, s^2 (s2) tends to be smaller than the true (population) variance.

2.2 Write Functions to Run Simulations

Above, we wrote R codes to run a simulation. However, using the codes above, we have to run several codes again and again to repeat simulations. To make it easy to iterate simulations, let's write a function to run simulations.

```
sim_var <- function(n, trials, true_var) {## function to simulate unbiased varia
    ## Aarguments:
    ##     n = sample size
    ##     trials = num of iterations in a simulation
    ##     true_var = sigma^2 (population variance)
    ## Rreturn: matrix of the means and sd's of s2 and u2
    s2 <- rep(NA, trials)
    u2 <- rep(NA, trials)
    for (i in 1:trials) { ## loop for a simulation
        x <- rnorm(n, sd = sqrt(true_var))  ## random sample from N(0, true.var)
        s2[i] <- sum((x - mean(x))^2) / n    ## sample variance
        u2[i] <- sum((x - mean(x))^2) / (n - 1) ## unbiased variance
    }
    res <- matrix(c(mean(s2), mean(u2), sd(s2), sd(u2)), nrow = 2)
    row.names(res) <- c("sample var.", "unbiased var.")
    colnames(res) <- c("mean", "sd")
    return(res)
}
```

Let's run a simulatio with this function. First, when $n = 5$, run:

```
sim_var(n = 5, trials = 1000, true_var = 10)
```

```
##                    mean       sd
## sample var.    7.971714 5.448018
## unbiased var.  9.964643 6.810023
```

We got the result.

Let's increase the smaple size to 10,

```
sim_var(n = 10, trials = 1000, true_var = 10)
```

```
##                    mean       sd
## sample var.    8.877485 4.281637
## unbiased var.  9.863872 4.757374
```

Using the function we created above, let's make a new function to run a simulation for different sample sizes simultaneously. Here, we define a function to simulate all n's between **n_min** and **n_max**.

```
sim_var2 <- function(n_min = 1, n_max, trials = 1000, true_var){
    ## Arguments:
    ##        n.min = the minimum, default to 1
    ##        n.max = the maximum, no default
    ##        trials = n. of iterations in a simulation, default to 1000
    ##        true_var = sigma^2 (the population variance)
    ## Return: A list of matrix returned by sim_var
```

```
    ## Print error message if the input value is wrong
    if (n_min < 1) stop(message = "n.min must be a positive integer")
    if (n_max < 1) stop(message = "n.max must be a positive integer")
    if (trials < 1) stop(message = "trials must be a positive integer")
    if (true_var < 0) stop(message = "true_var must be a positive value")

    ## sample sizes
    n_vec <- n_min:n_max
    ## matrix to save the result
    output <- matrix(NA, nrow = length(n_vec), ncol = 5)
    colnames(output) <- c("n", "s2_mean", "u2_mean", "s2_sd", "u2_sd")
    ## run simulation by for loop
    for (i in seq_along(n_vec)) {
        ## use the function we made before
        ## save the i-th result in the i-th row
        output[i, 1] <- n_vec[i]
        output[i, 2:5] <- as.vector(sim_var(n = n_vec[i], trials = trials,
                                            true_var = true_var))
    }
    return(output)
}
```

For instance, run a simulation for $n = 10, 11, \ldots, 100$, and visualize the results.

```
sim1 <- sim_var2(n_min = 10, n_max = 100, trials = 1000, true_var = 10)
```

Lastly, let's visualize the result.

```
## make a data frame from the matrix
df <- data.frame(n = rep(sim1[, 1], 2),
                 mean = c(sim1[, 2], sim1[, 3]),
                 sd = c(sim1[, 4], sim1[, 5]),
                 type = c(rep("biased", dim(sim1)[1]), rep("unbiased", dim(sim1)
df <- df %>%
    mutate(lb = mean - sd, ub = mean + sd)
## make a plot
res_sim1 <- ggplot(df, aes(x = n, y = mean, color = type))
res_sim1 +
    geom_line() +
    geom_abline(intercept = 10, slope = 0, color = "royalblue", linetype = "dash
    scale_color_discrete(name = "variance", labels = c(expression(s^2), expressi
```

The next figure displays the same result with error bars (mean ± sd).

```
last_plot() +
    geom_pointrange(aes(ymin = lb, ymax = ub))
```

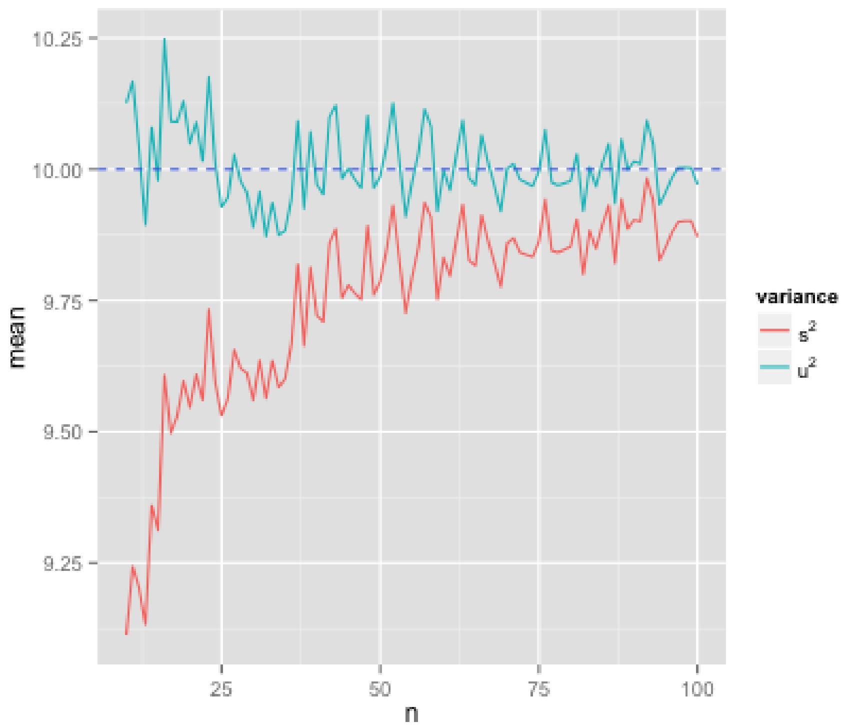

Figure 2: plot of chunk visualize-simulation

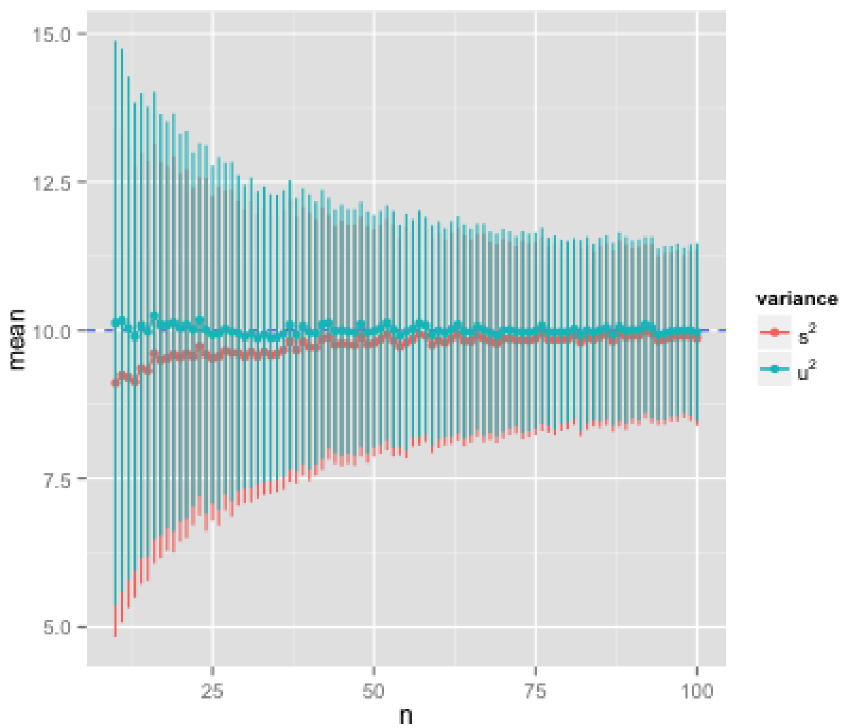

Figure 3: plot of chunk visualize-simulation-errors

Back to Class Materials