政治学方法論 I

R マークダウンを用いた文芸的プログラミング入門

矢内　勇生

2016 年 *5* 月 *22* 日

目次

準備

R マークダウン (R Markdown、拡張子は.Rmd) ファイルはテキストエディタや RStudio をはじめとする IDE で編集する。編集したファイルを HTML や PDF に出力するために、`rmarkdown::render()` や `knitr::knit()` を利用する。また、日本語を含む図の出力には **Cario** パッケージを使う。これらがインストール済みでない場合はインストールし、パッケージを読み込む。

```
if (!require(rmarkdown)) {
    install.packages('rmarkdown', dependencies = TRUE)
    library('rmarkdown')
}
if (!require(knitr)) install.packages('knitr', dependencies = TRUE)
# 開発版もインストールする
if (!require(devtools)) install.packages('devtools', dependencies = TRUE)
devtools::install_github('yihui/knitr')
library('knitr')
if (!require(Cairo)) {
    install.packages('Cairo', dependencies = TRUE)
    library('Cairo')
}
```

R マークダウンの使い方

マークダウンファイル (literate-programming.Rmd) とそのファイルを元に生成された html ファイル (lieterate-programming.html) や PDF ファイル（literate-programming.pdf）を見比べながら、RStudio で R マークダウンファイルを扱えるようにするのが今日の目標である。

このマークダウンをそのまま使うためには、担当教員が作ったスタイルシート（my-markdown.css）をホームディレクトリに保存する必要がある。スタイルをカスタマイズしたいなら、このファイルを変更すればよい。デフォルトのスタイルのままでいいとき（あまり良くないと思うが）は、ヘッダの'css' オプションの指定をやめる（この Rmd ファイル [html ではない] のヘッダ部分にある "css: my-markdown.css" の行を削除する）。

マークダウン記法を利用した文書の書き方

文章は、いつもどおり書けばよい。文章の一部をイタリックにしたいときは *this is italic* あるいは *this is also italic* とする。 太字は、**ここが太字** または**ここも太字** とする。 太字のイタリックは、***here is bold italic*** または ***here is also bold italic*** とする。

改行するときは、文章の間を 1 行以上空ける。

箇条書きは、

- 項目 1
- 項目 2
  - 項目 2-1
  - 項目 2-2

あるいは、

- 項目 1
  - 項目 1-1
  - 項目 1-2
- 項目 2

のようにする。* や - の後は半角スペースを空ける。箇条書きを入れ子ににするとき、字下げは Tab で行う

番号付きの箇条書きは、

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

のようにする。

また、リンクを貼ることもできる：矢内のウェブサイト。

画像も貼れる：  （画像ファイルは、ココにある）

数式の書き方

LaTeX と同じように数式を書くこともできる。文章中と同じ行に数式を書きたいときは、$で挟む。たとえば、$\bar{x} = \sum_{i=1}^{n} x_i/n$ と、する。数式を独立したブロックとして書きたいときは、$$で挟み、

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

のようにする。

コードチャンクの書き方

R のコードは、コードチャンクと呼ばれる部分に書き込む。コードチャンクは、たとえば以下のように書ける。

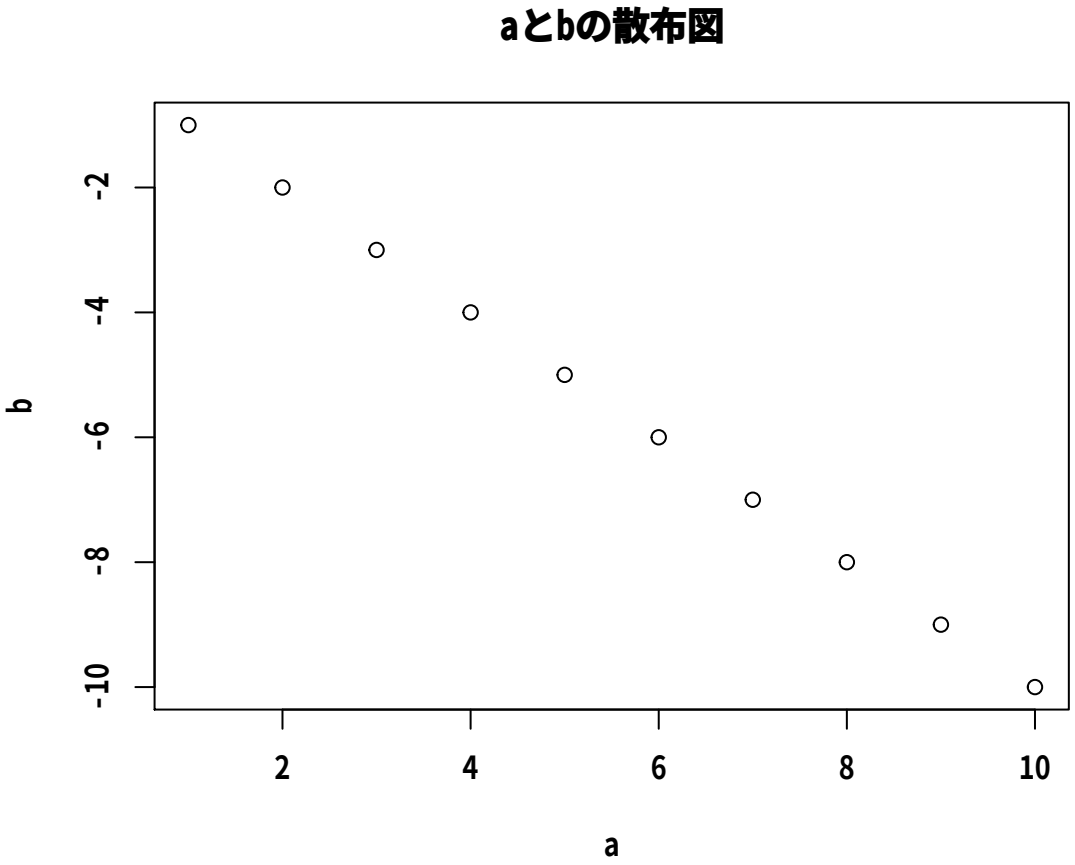
```
a <- 1:10
b <- -1:-10
```

R コードチャンクの始めには、3 つの「‘」の後に{r}をつける。r とスペースの後（{}の中）には、チャンクの名前を付ける。好きな名前を付けてよいが、他のチャンクとまったく同じ名前は付けられない。チャンクの終わりには 3 つの「‘」を書く。

文章中に R コードを書きたいときは `mean(x)` のように書く。関数を実行（評価, evaluate）した後の結果を文章中に入れたいときは、「a の平均値は 5.5 です」のように “r” を入れて書く。この方法を使えば、シミュレーションなどで結果が変わっても文章を書き直す必要がない。

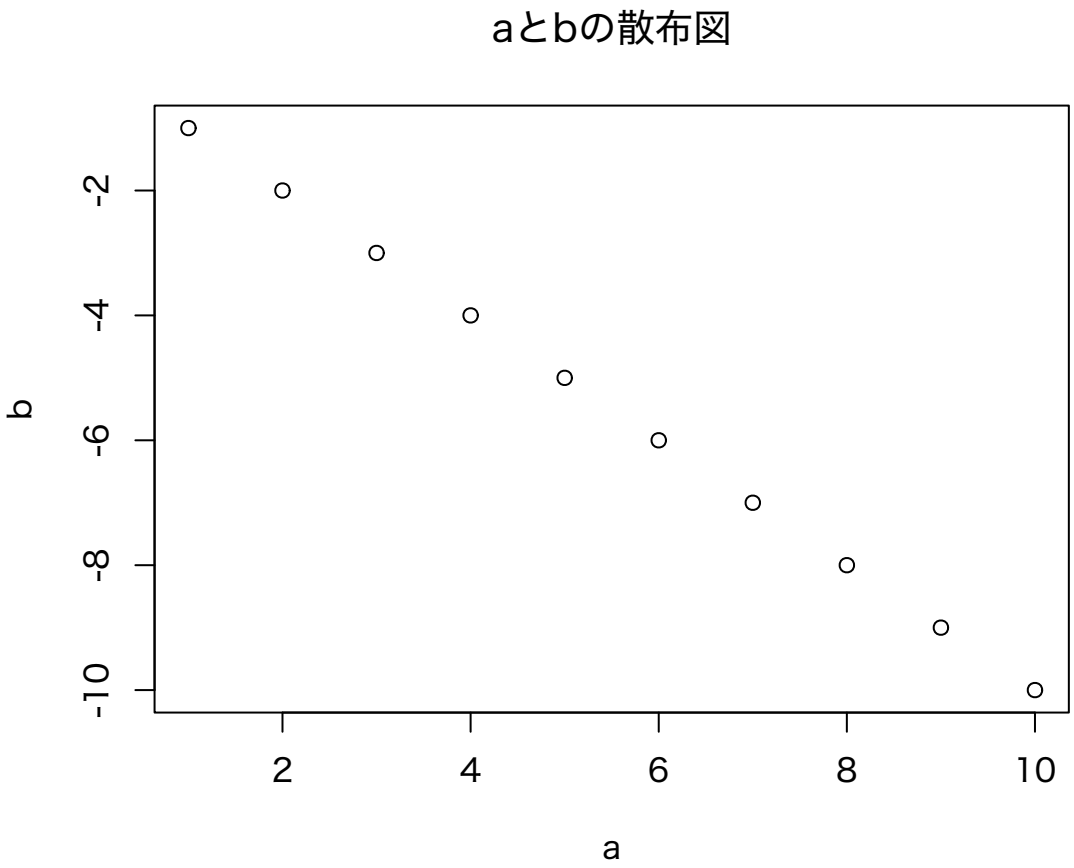
図を含めた文章も作れる。

```
plot(a, b, main = 'a と b の散布図')
```



日本語を含む図を PDF に出力したいときは、オプションで **cairo_pdf** をグラフィックデバイスに指定し、`par(family = ...)` で日本語が表示できるフォントを選ぶ。（PDF 用なので、HTML 上では適切に表示されない）

```
# フォントは利用可能なものの中から好きなものを選ぶ
par(family = 'Hiragino Sans')  # Mac 用
# par(family = 'Meiryo')  # Windows 用
plot(a, b, main = 'a と b の散布図')
```



この方法だと、図のタイトルが図の上にできてしまう。図のキャプションは下にある方がいいので、次のようにする。

```
# フォントを欧文フォントに戻す
par(family = 'Helvetica')  # Mac
# par(family = 'Arial')  # Windows
plot(a, b, main = '')
```

何もオプションを指定しない状態では、チャンクは 1 行ずつ評価され、結果も順番に次々出力される。たとえば、

```
sd(a)
```

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

```
var(a)
```

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

チャンクの最後まで評価してからまとめて結果表示したいときは、チャンクオプション **results** を'hold' にする。オプションは、チャンク名の後に「,（comma）」を打ち、その後に書く。

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

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

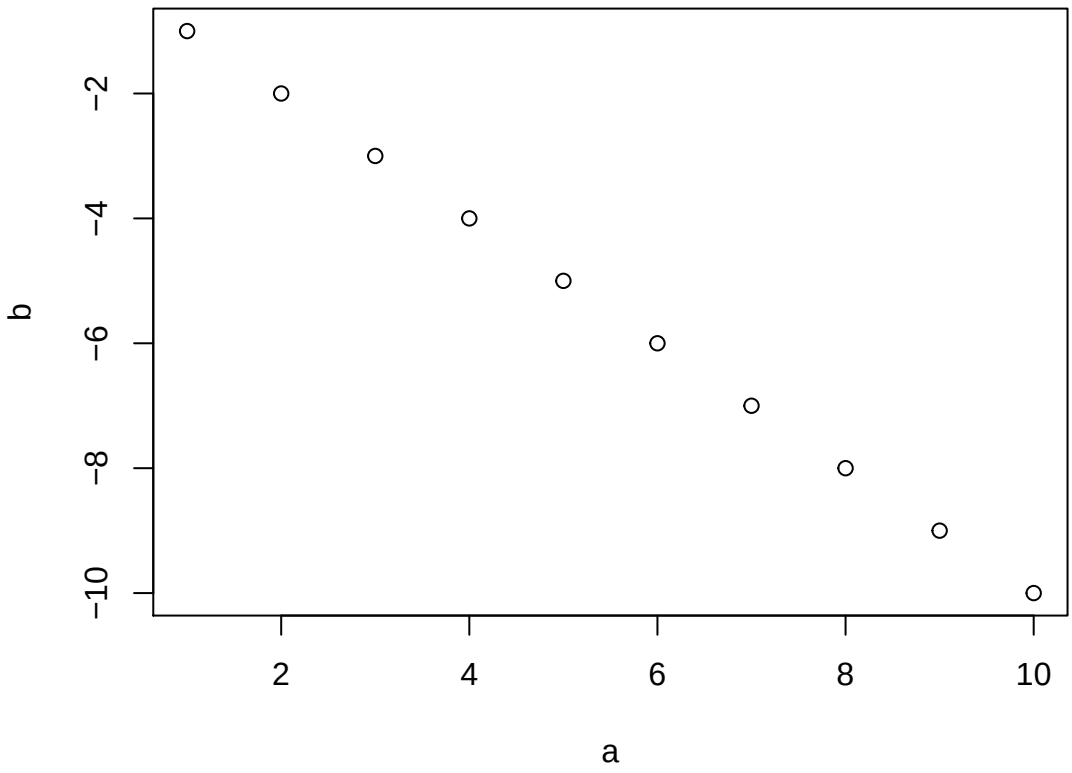


図 1　a と b の散布図

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

チャンクオプションについてより詳しくはココなどを参照されたい。

また、R マークダウン全般（特に、RStudio を使う場合）については、ココを参照。

R Markdown ファイルを他のファイル形式に出力する

R Makdown から HTML へ

R Markdown を HTM ファイルに出力するときは、`rmarkdown::render()` を使う。

HTML ファイルに変換する際のオプションは、ヘッダ部分で指定する。この Rmd ファイルでは、第 1 行から第 18 行の間がヘッダであり、そのうち、“html_document:” のブロックで HTML 出力のためのオプションが指定されている。例えば、“toc: yes” は目次 (table of contents; toc) を表示するという指定である。非表示にするには “toc: no” とする。

試しに、“literate-programming.Rmd” を “literate-programming.html” に変換してみよう。 Rmd ファイルを RStudio で編集している場合、コード編集画面の上にある “Knit HTML” ボタンを押しても HTML ファイルを作る。

出力された HTML ファイルは（他のディレクトリを指定しない限り）現在の作業ディレクトリに保存される。

出来上がった HTML ファイルをブラウザで開いて確認してみよう。

コマンドを使って HTML ファイルを作るときは、`rmarkdown::render()` を使う。

```
render('literate-programming.Rmd', output_format = 'html_document',
       output_file = 'literate-programming.html', run_pandoc = FALSE)
```

run_pandoc = FALSE を指定しないと日本語が文字化けするので注意が必要である。

R Markdown から PDF へ

R マークダウンファイルから PDF ファイルに出力することも可能である。PDF に出力する際のオプションは、ヘッダの “pdf_document:” のブロックで指定されている。ただし、PDF 出力には **LaTeX** と **Pandoc** が必要である。

LaTeX と Pandoc が使えるなら、“Knit PDF” ボタン（“Knit HTML” ボタンの右にあるプルダウンメニューから選択）をクリックすれば PDF ファイルができる。現在の作業ディレクトリにこのような PDF ファイルができるはずである。

コマンドで PDF を作るときは、次のようにする。

```
render('literate-programming.Rmd', output_format = 'pdf_document',
       output_file = 'literate-programming.pdf')
```

R Markdown の例：R によるシミュレーション

分散と不偏分散

確率変数 X の母分散が σ^2 だとする。このとき、X の標本分散を

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

とすると、

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

となる。つまり、s^2 は σ^2 の不偏推定量ではない。代わりに、

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

と定義すると、 この u^2 が σ^2 の不偏推定量になる。このことは簡単に証明できるが、ここでは R でシミュレーションを行うことによって確認してみよう。

シミュレーション条件の設定

まず、シミュレーションの条件として、サンプルサイズ (n)、シミュレーションの繰り返し回数 (trials)、真の母分散の値 (σ^2) を決める。また、シミュレーションの結果を保存するための変数を用意する。

```
n <- 10
trials <- 1000
sigma2 <- 10    # True Variance
```

```r
# prepare vectors to save the results
s2 <- rep(NA, trials)
u2 <- rep(NA, trials)
```

ここまでで準備ができたので、実際にシミュレーションを行う。ここでは、for ループを利用する。

```r
for (i in 1:trials) {  ## シミュレーションを実行するループ
    x <- rnorm(n, sd = sqrt(sigma2))    # N(0, sigma^2) からの無作為抽出
    s2[i] <- sum((x - mean(x))^2) / n  # 標本分散の計算
    u2[i] <- sum((x - mean(x))^2) / (n - 1)  # 不偏分散の計算
}
```

シミュレーションが終わったので、結果を確認してみよう。

```r
# variance
mean(s2)  # 標本分散の平均値
```

```
## [1] 9.026181
```

```r
mean(u2)  # 不偏分散の平均値
```

```
## [1] 10.02909
```

このように、s^2 (s2) は平均すると真の値よりも小さめの値を出してしまうことがわかる。

関数を定義してシミュレーションを効率的に行う

上の例では、シミュレーションの条件を変更するたびに複数のコードを実行し直す必要があって不便である。そこで、シミュレーションを 1 行でやり直せるよう、シミュレーション用の関数を定義する。

```r
sim_var <- function(n, trials, true_var) {# 不偏分散をシミュレートする関数
    ## 引数 (arguments):
    ##    n = 1 つのサンプルのサイズ
    ##    trials = シミュレーションの繰り返し回数
    ##    true_var = sigma^2 (真の母平均)
    ## 返り値 (return): s2 と u2 の平均と標準偏差 (行列)
    s2 <- rep(NA, trials)
    u2 <- rep(NA, trials)
    for (i in 1:trials) { # シミュレーションを実行するループ
        x <- rnorm(n, sd = sqrt(true_var))  ## N(0, true.var) からの無作為抽出
        s2[i] <- sum((x - mean(x))^2) / n   ## 標本分散を計算する
        u2[i] <- sum((x - mean(x))^2) / (n - 1) ## 不偏分散を計算する
    }
    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)
}
```

これで関数が定義できた。

この関数を使ってシミュレーションを行ってみよう。 まず、$n = 5$ で 1,000 回繰り返してみる。

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

```
##                   mean       sd
## sample var   7.982188 5.791397
## unbiased var 9.977735 7.239247
```

これでシミュレーションの結果が出た。

n の値を 10 に変えると、

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

```
##                   mean       sd
## sample var   8.826617 3.908636
## unbiased var 9.807352 4.342929
```

となる。

この関数を使い、複数の n についてのシミュレーションを一挙に行う関数を新たに定義しよう。ここでは、シミュレーションを行いたいうちで最も小さな n の値 **n_min** と最も大きな値 **n_max** を与えたとき、その間（端点を含む）にあるすべての n についてシミュレーションを実施する関数を作る。

```
sim_var2 <- function(n_min = 1, n_max, trials = 1000, true_var){
    ## 引数：n_min = 最も小さな n，既定値は 1
    ##       n_max = 最も大きな n
    ##       trials = シミュレーションの繰り返し回数，既定値は 1000
    ##       true_var = sigma^2 （真の母平均）
    ## 返り値：行列（行数 = n の数，列数 = 3）

    # 入力がおかしいときはエラーを出す
    if (n_min < 1) stop('n_min には 1 以上の整数を指定してください')
    if (n_max < 1) stop('n_max には 1 以上の整数を指定してください')
    if (trials < 1) stop('trials には 1 以上の整数を指定してください')
    if (true_var < 0) stop('true_var には 0 以上の実数を指定してください')

    # シミュレーションに利用する n
    n_vec <- n_min:n_max
    # 結果を保存するための行列
    output <- matrix(NA, ncol = 5, nrow = length(n_vec))
    colnames(output) <- c('n', 's2_mean', 'u2_mean', 's2_sd', 'u2_sd')
    # ループを使ってシミュレーションを実行する
    for (i in seq_along(n_vec)) {
        # 既に定義済みの関数を利用する
        output[i, 1] <- n_vec[i]
        output[i, 2:5] <- sim_var(n = n_vec[i],
                                  trials = trials, true_var = true_var)
```

```
    }
    return(output)
}
```

これで、関数ができた。

ためしに、$n = 10, 11, \ldots, 100$ について、この関数を使ってシミュレーションを実行し、結果を図示してみよう。

```
library('ggplot2')
library('dplyr')
sim1 <- sim_var2(n_min = 10, n_max = 100, trials = 1000, true_var = 10)
df <- data.frame(n = rep(sim1[, 1], 2),
                 mean = c(sim1[, 2], sim1[, 3]),
                 sd = c(sim1[, 4], sim1[, 5]))
df <- df %>%
    mutate(type = c(rep(' 標本分散', n()/2), rep(' 標本不偏分散', n()/2)),
           lb = mean - sd, ub = mean + sd)
res_sim <- ggplot(df, aes(x = n, y = mean, ymin = lb, ymax = ub)) +
    facet_grid( . ~ type) +
    geom_ribbon(fill = 'lightblue') +
    geom_hline(yintercept = 10, color = 'firebrick') +
    geom_line(color = 'darkblue') +
    ylab(' 分散')
```

```
# PDF にはこのチャンクを使う
print(res_sim + theme_gray(base_family = 'Hiragino Sans'))  # Mac
```

```
#print(res_sim + theme_gray(base_family = 'Meiryo'))  # Windows
```

```
# HTML にはこのチャンクを使う
#print(res_sim + theme_gray(base_family = 'Osaka'))  # Mac
#print(res_sim + theme_gray(base_family = 'Meiryo'))  # Windows
```

このように、偏差平方和を標本数で割ると、分散を過小推定する傾向があることがわかる。

授業の内容に戻る

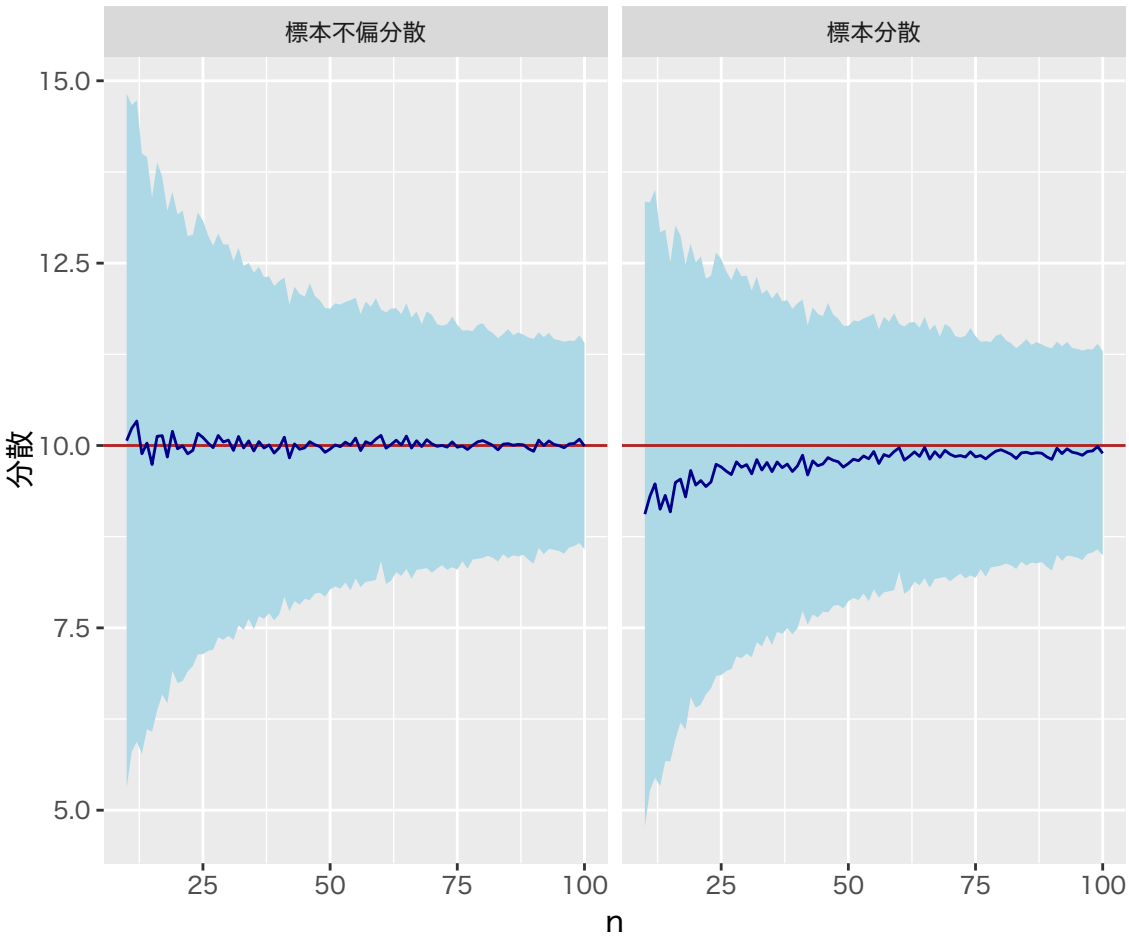


図 2　不偏分散のシミュレーション