R でシミュレーション

おさらい – 関数 -

シミュレーションを行う前に，まず関数のおさらいをしましょう．

R にはたくさんの関数が存在します．
それぞれ使用する際には，決められた変数に任意の引数を入力します．

例えば，与えたベクトルの平均値を知りたいとき，関数 mean()を使い，変数に**”x”というベクトルを引数として入力**しますね．

```
> x <- c(1, 5, 6, 8, 3, 2, 7)
> mean(x)
[1] 4.571429
```

さて、ではより複雑な処理を繰り返し行いたい場合，どうすればいいのでしょう…？

自分で関数をつくってしまえばいいのです

R では関数 function()を使うことで好きな関数をつくることができます．
用意する値は**”関数の名前”**，関数内の計算で使用する**“変数”**，**“計算式”**．
次のような形で入力します．

```
名前 <- function(変数 1, 変数 2, …, 変数 n) {
  計算式
    :
  return(返したい値)
}
```

例えば，変数 x と y に関して、2 つの和を求める関数をつくるとこのようになります．

```
hello <- function(x, y) {
  a <- x + y
  return(a)
}
```

実際に値を入れてみます．

```
> hello(2, 7)
[1] 9
```

ここで大事なことが 1 つ．関数を定義したときに R が記憶するのは，**”hello という関数がある”**ということだけです．
関数内で設定した”a”に関しては記憶してくれないので，常に使う値があるならば外に出しておきましょう．
逆に，仮に関数の外で”a”に何かを代入していたとしても，それぞれの値は全くの別物として扱われるので，関数内では外の値と**重複して設定しても問題ありません**．

乱数の発生

シミュレーションを行うためには，乱数を発生させる必要があります．
乱数とは，ある確率分布に沿った値を無作為に抽出したものです．

一様分布に従う乱数を出力する関数 runif()は，よく用いられる一般的な関数です．使い方は非常に簡単で，欲しい乱数の値と，その上限下限を入力するだけです．
例えば，0 から 100 までの一様な乱数を 5 個発生させるとします．

```
> runif(5, min = 0, max = 100)
[1] 21.09087 12.01319 72.65015 25.10465 38.01219
```

シードを設定することで，複数回試行を行った場合でも常に同じ値を得ることができます．

```
> set.seed(1)
> runif(5, min = 0, max = 100)
[1] 26.55087 37.21239 57.28534 90.82078
[5] 20.16819
> set.seed(1)
> runif(5, min = 0, max = 100)
[1] 26.55087 37.21239 57.28534 90.82078
[5] 20.16819
```

また，関数 as.integer()を利用することで整数値での出力も可能です．

```
> set.seed(1)
> as.integer(runif(5, min = 0, max = 100))
[1] 26 37 57 90 20
```

これを利用すると，シミュレーションを行うことができます．

シミュレーション

例えば，1 つの箱に 1 から 100 までの番号が書かれたボールが入っているとします．この中から 5 つ取り出して足し合わせ，また箱の中へ戻すということを 10000 回やろうと思います．ところがこれを手でやろうとすると相当な時間と労力がかかります．
そこで，値を入力すると**「1 から 100 までの値から，5 つの乱数を発生させ足し合わせる工程を，10000 回行う」**という試行について，lattice というパッケージを使って**密度推定曲線**を描き出してみようと思います．

```
library(lattice)

DEN <- function(n, Min, Max, freq) {
    y <- c()
    for (i in 1:freq) {
    x <- as.integer(runif(n, min = Min, max = Max))
    y <- c(y, sum(x))
    }
  densityplot(y, col = "deeppink", xlab = "")
}
```

設定した変数は以下の通り
n: 発生させる乱数の数
Min: 乱数の下限
Max: 乱数の上限
freq: 試行を繰り返す回数

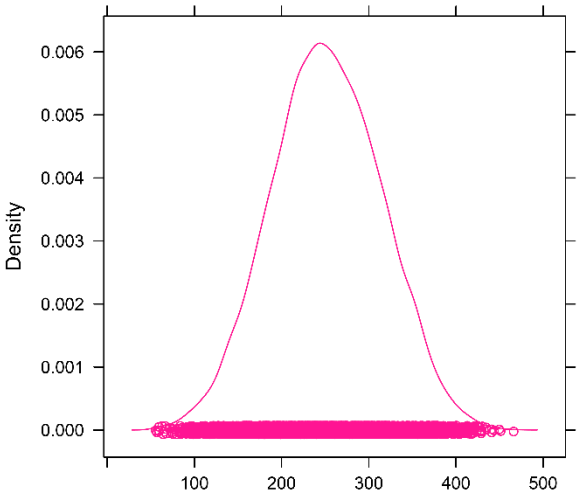
グラフの色や凡例の内容を変数に設定するのも良いかもしれません．

上でも記述したように，関数内での出来事は関数内でしか記憶されません．そのため，使う**パッケージの読み込み**は，**関数の外**に出しておいた方が無難です．より複雑な処理をさせようとした時に毎回のようにパッケージを読み込んでいては，作業が重くなって進まなくな

る原因になりかねません．少しでも**軽い処理**にすることがポイントです．

実際にコードを入力するとこのような結果が得られます．

```
> DEN(5, 1, 100, 10000)
```



これは基本のシミュレーションの方法ですが，他にも様々な試行の方法があります．
ぜひ自分のニーズに合ったシミュレーション法を試してみてください．