hw03j: ベクトルと行列

Kenji Sato*

2017/03/30

1 概要

目的

R でベクトルと行列を扱えるようになる。ユーザー定義関数の書き方を学ぶ。

課題

- レポジトリのクローンし，作業ブランチを作成する（eg. `solution` ブランチ）
- `problem.pdf` を読み指示に従って，`R/matrix.R` に第 5 節の問題に対する解答を書く。
- `solution.Rmd` を knit して `solution.pdf` を作成する
- `R/matrix.R`, `solution.Rmd` と `solution.pdf` をコミットする
- プルリクエストを送信する

2 R クイックコース

2.1 アトミックベクタ

数ベクトルは数字の組である。数学で次のような式を見たときには，3 次元数ベクトル空間のベクトルであると理解する。

$$x = \begin{bmatrix} 2.1 \\ 1.2 \\ 4.3 \end{bmatrix}$$

これらは，$\mathbb{R}^3$, $\mathbb{C}^3$ か $\mathbb{Q}^3$ の元である。

R では関数 `c()` を使って数字をつなぐ。

*神戸大学. Email: mail@kenjisato.jp

```
x <- c(2.1, 1.2, 4.3)
x
```

```
## [1] 2.1 1.2 4.3
```

x のようなオブジェクト（変数）は R ではアトミックベクタと呼ばれる。すべての要素が共通の型（x の場合は numeric）を持ち，一列に並んでいる。

アトミックベクタから数値を読み出すには，添字関数 [を使う[1]。 R ではインデックスは 1 から始まる[2]。

```
x[1]
```

```
## [1] 2.1
```

同様に

```
x[2]
```

```
## [1] 1.2
```

添字としてベクトルを使うこともできる。

```
x[c(2, 3)]
```

```
## [1] 1.2 4.3
```

負の添字も使える。

```
x[-1]
```

```
## [1] 1.2 4.3
```

負の添字を複数使うこともできるが，正と負を同時に使うことはできない。

```
x[c(1, -2)]
```

```
## Error in x[c(1, -2)]: only 0's may be mixed with negative subscripts
```

2.2 スカラー

R にはスカラーはない。単一の数字はサイズが 1 のベクトルである。

```
1 == c(1)
```

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

次の振る舞いに惑わされないように。

[1] [は関数である。例えば，次のような関数呼び出しを試してほしい `[`(x, 1)。
[2] アトミックベクタ x の n 番目の要素は小数部分が n になるように定義した。このアイデアは Hadley Wickham http://adv-r.had.co.nz/Subsetting.html を参考にした。

```
length(0)
```

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

2.3 計算

同じサイズのベクトル同士の加算と減算は期待通りの振る舞いである。

```
y <- c(-2, -1, -4)
x + y
```

```
## [1] 0.1 0.2 0.3
```

積と除算は要素ごとに計算される。

```
x * y
```

```
## [1]  -4.2  -1.2 -17.2
```

2.4 列ベクトル・行ベクトル

数学的には

$$x = \begin{bmatrix} 2.1 \\ 1.2 \\ 4.3 \end{bmatrix}$$

と

$$x = \begin{bmatrix} 2.1 & 1.2 & 4.3 \end{bmatrix}$$

を異なるものと考えることも多い。前者を列ベクトルと呼び，後者を行ベクトルと呼ぶ。`c(2.1, 1.2, 4.3)` には行・列の区別はない。

列ベクトルを作るには，`matrix()` 関数を使う。

```
xcol <- matrix(x)
xcol
```

```
##      [,1]
## [1,]  2.1
## [2,]  1.2
## [3,]  4.3
```

行ベクトルを作る場合にも同様に，

```
xrow <- matrix(x, nrow = 1)
xrow
```

```
##      [,1] [,2] [,3]
## [1,]  2.1  1.2  4.3
```

違った方法としては，列ベクトルを転置して行ベクトルを作ることもできる。転置は t() 関数を使う。

```
t(xcol)
```

```
##      [,1] [,2] [,3]
## [1,]  2.1  1.2  4.3
```

行・列を指定すると，通常の演算に制限が加わる。

```
xcol * xrow
```

```
## Error in xcol * xrow: non-conformable arrays
```

```
xrow * xcol
```

```
## Error in xrow * xcol: non-conformable arrays
```

行列積を計算する場合には %*% 演算子を使う。

```
xrow %*% xcol
```

```
##       [,1]
## [1,] 24.34
```

2.5 ユークリッドノルム

ベクトルの長さ（ノルム）を計算するのは，定義に従って次のようにすればよい。

```
sqrt(sum(x ^ 2))
```

```
## [1] 4.933559
```

これは，列・行ベクトルにも使用できる。

```
sqrt(sum(xrow ^ 2))
```

```
## [1] 4.933559
```

2.6 行列

列・行ベクトルはそれぞれ列・行を 1 つだけもつ行列に過ぎない。一般の行列も同様に定義できることは，想像に難くないだろう。（なにせ matrix() 関数を使っている）

```
elm_colwise <- c(1.11, 3.21, -5.31, 2.12, -6.22, 0.32)
matrix(elm_colwise, nrow = 3)
```

```
##       [,1]  [,2]
## [1,]  1.11  2.12
## [2,]  3.21 -6.22
## [3,] -5.31  0.32
```

デフォルトでは matrix() 関数は要素を列方向に埋めていく。行方向に埋めたい場合は，byrow = TRUE をパラメータとして渡す。

```
elm_rowwise <- c(
     1.11,  2.12,
     3.21, -6.22,
    -5.31,  0.32
  )
matrix(elm_rowwise, nrow = 3, byrow = TRUE)
```

```
##       [,1]  [,2]
## [1,]  1.11  2.12
## [2,]  3.21 -6.22
## [3,] -5.31  0.32
```

通常の演算は，サイズが揃っていれば計算できる。

```
set.seed(1)
m1 <- matrix(rnorm(9), nrow = 3)
m2 <- matrix(rnorm(9), nrow = 3)
```

加算

```
m1 + m2
```

```
##            [,1]       [,2]      [,3]
## [1,] -0.9318422  0.9740402 0.4424954
## [2,]  1.6954245 -1.8851921 0.7221344
## [3,] -0.4457854  0.3044625 1.5196176
```

減算

```
m1 - m2
```

```
##            [,1]      [,2]       [,3]
## [1,] -0.3210654  2.216521  0.5323627
## [2,] -1.3281378  2.544208  0.7545150
## [3,] -1.2254718 -1.945399 -0.3680549
```

積。* ではないことに注意。

```
m1 %*% m2
```

```
##            [,1]        [,2]      [,3]
## [1,]  2.7930481 -2.59556567 0.4623740
```

```
## [2,]  0.7298920 -0.01328322 0.6832710
## [3,] -0.7607129  2.98393189 0.5942747
```

2.7 比較

非整数値が等しいことを比較するのに == を使ってはいけない。

次のコードには問題はない

```
1 == 2 - 1
```

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

しかし，次のコードには重大な問題がある。

```
0.3 == 0.1 + 0.1 + 0.1
```

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

小数（正確には浮動小数点数）が等しいまたは近いことをチェックするには all.equal() を使う。

```
all.equal(0.3, 0.1 + 0.1 + 0.1)
```

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

この関数は，ベクトルや行列にも同様に使うことができる。

```
all.equal(m1, m2) == TRUE
```

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

3 練習問題 1: 関数定義

似たようなコードを繰り返し書いていることに気がついたら，関数を定義することを検討しよう。メンテナンスが容易になるし，バグの可能性を減らすことができる。

練習として，上で計算したノルムを計算する関数 norm() を定義しよう。次のような呼び出しでノルムが出力されると非常にコードが読みやすくなるだろう。

```
norm(x)  #> Should return 4.933559
```

解答

R では次の様に関数を定義する。

```
norm <- function(x) {
  sqrt(sum(x ^ 2))
}
```

4 練習問題 2: 外部ファイルを source() する

いつも Rmd ファイルに関数を書き始めるのはお薦めできない。関数を定義する理由は使い回しをしたいからなのに，Rmd ファイルにしか定義がないと他のレポートで使うことが難しい。それに Rmarkdown のコンパイルにはやや時間がかかるので，実験段階ではもっと計量なフォーマットで作業を行うのがよいだろう。拡張子.R を付けたスクリプトファイルに書くのが普通である。

Files ペインから R/vector.R を見つけて，開けてみてほしい。そこに norm() 関数の定義が書かれている。この関数を Rmarkdown ファイルで使うには source() すればよい。

```
source('R/vector.R')
```

その後，

```
norm(x)
```

```
## [1] 4.933559
```

プロジェクトで使われるすべての .R ファイルを R という名前のフォルダに入れることをお薦めする。R の中にはサブフォルダを作らない[3]。

[3] R パッケージを書く際にはこのルールに従わなければならない

5 問題

2 つのファイルを編集する。

- `solution.Rmd`
  - 名前を書く
- `matrix.R`
  - 関数 `is_symmetric()` を定義する。仕様は以下を参照。

そして `solution.Rmd` を knit する。生成された PDF にエラーや警告がなければ終了。コミットしてプルリクエストを送る。

仕様

名前：`is_symmetric()`

- 正方行列 `x` を 1 つパラメータとして受け取る
- `x` がスカラーなら `TRUE` を返す
- `x` が対称なら `TRUE`，`x` が非対称なら `FALSE` を返す

`x` が正方行列であるかどうかのチェックはしなくてもよいが，余力があればチェックをするようにしてもよい。この関数は `base::isSymmetric()` に似ているが，`base::isSymmetric()` はスカラーに対して適用できない。