「R入門」

紹介と準備
簡単な操作

4月8日（火）

新納浩幸

Rとは

データ操作，計算，グラフ表示のソフト

→ 一般的には統計解析ソフトとして利用

特徴

・統計解析ソフトSとほぼ同じ操作体系と処理が可能

世界最高の統計解析ソフト

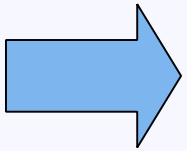
・フリー

プログラム言語としてのR

プログラム言語の1つと捉えた方がわかりやすい

各々のプログラム言語は対象とする分野が存在する

C: 制御用
Fortran: 数値計算用
Lisp: 記号処理用
Java: ネットワーク用

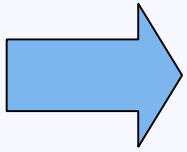


R: データ解析用
データをひとつのオブジェクトとみなし,
そのオブジェクトに対して一連の処理が可能

プログラム言語が動く環境

あるプログラム言語で作成されたプログラムは
ある環境で動く

C : Cコンパイラーが動く環境の上で動く
Lisp: Lisp 環境の上で動く
Prolog: Prolog 環境の上で動く
Bash: UNIX 環境の上で動く
VBA: Office の上で動く
etc



R : Rの環境の上で動く

最低これだけは知らないとダメ

Rはインタプリンタ

(1) 対話的な操作
(2) 変数の型宣言などない
(3) オブジェクトを作り、名前をつける（変数化する）
(4) オブジェクトへは名前を介してアクセスできる
(5) 変数名（＋リターン）でそのオブジェクトが表示される
(6) 変数化されたオブジェクトは明示的に消さなかぎり残り、
環境内でいつでも参照可能

R（R環境）の起動と終了

問題ごとに作業ディレクトリを作る

起動はディスクトップのアイコンをクリック

終了は R 環境の中で

```
> q()
```

環境を保存するかどうか聞かれる．通常保存する．

ヘルプ

R環境内でマニュアルを参照できる

> help(関数名)

例）> help(max)

> help.start()　　###　マニュアルがブラウザで見れる

R環境内で使用例を調べられる

> example(トピック名)

例）> example(max)

大文字と小文字の区別等

＊Rでは大文字と小文字は区別される

＊変数名、関数名は英数字からなる

. (ドット)も使える
数字で始まってはダメ
_ (アンダーバー)は使ってはダメ

＊コメントは　＃
＊文は ;　か改行(; を使って複数の文が1行に書ける)
＊{} で複数の文を1文にまとめられる

命令の編集

Emacs ライクな操作、、、

カスタマイズも可能（らしい）

ファイルから/への入出力

入力

例）commads.r として以下の内容のファイルを作成する

```
x ← 1:10
y ← mean(x)
```

```
> source(“commands.r”)
> y
[1] 5.5
```

出力

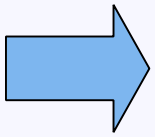
```
> sink(“kekka”)
> y
> sink()
```

出力をファイル kekka へ

出力をR環境内へ

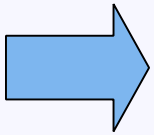
データの保存と削除

R終了時にデータを保存した場合



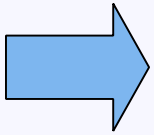
R環境内で作ったデータや命令の履歴が
.Rdata と .Rhistroy に保存される。
次にRを起動すると自動的に終了時の状態になる

どのようなデータが残っているかを見るには？



> objects()

データを削除するには



> rm(オブジェクト名)

ベクトルと付値

5次元ベクトル x =（10.4, 5.6, 3.1, 6.4, 21.7）の実現

x ← c(10.4, 5.6, 3.1, 6.4, 21.7)

関数 c が基本

以下の使い方も注意

y ← c(x,0,x)

y =（ 10.4, 5.6, 3.1, 6.4, 21.7, 0, 10.4, 5.6, 3.1, 6.4, 21.7 ）
　　x　　　　　　　　　　　　　　x

となる

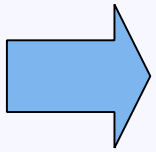
ベクトル演算（1）

+, -, log, exp, sin, cos, tan, sqrt, max, min, sum, prod, length, var .....

各々の関数の働きは概ね予想通り

1次元の関数はベクトルの各要素に作用する

x ← c(1, 3, 5, 9)
y ← sqrt(x)

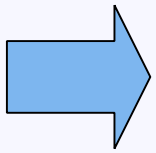


y = (1.00, 1.73, 2.23, 3.00)

ベクトル演算(2)

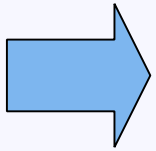
次元数の違うベクトルの演算は必要な分だリサイクルされる

```
x ← c(1, 2)
y ← c(3,4,5,6,7)
z ← x+y
```



z = (4, 6, 6, 8, 8)

```
x ← c(1, 2)
y ← c(3,4,5,6,7)
z ← x+y-1
```

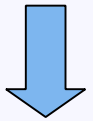


z = (3, 5, 5, 7, 7)

ベクトル演算（3）

行列に対してどのような結果が返るかは注意
特に、、、var()

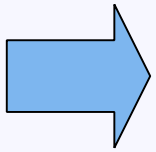
x が n 行 p 列の行列の時、var(x) は p 行 p 列の共分散行列になる



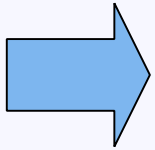
「パターン認識」で頻出している
忘れている人はチェック！！！

ただし標本に基づいているので、
サンプル数はマイナス1されることに注意

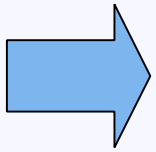
規則的な数列の生成

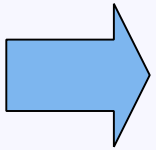
x ← 3:10  x = (3, 4, 5, 6, 7, 8, 9, 10)

コロン演算子は優先順位が最も高い

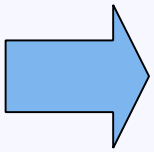
x ← 2*3:10  x = (6, 8, 10, 12, 14, 16, 18, 20)

関数 seq() の利用

x ← seq(2,10)  x ← 2:10　と同じ

x ← seq(2,10, by=3)  x = (2, 5, 8)

関数 rep() の利用

x ← 2:4
y ← rep(x,times=3) 

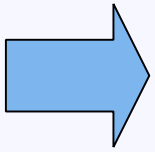
x = (2, 3, 4)
y = (2, 3, 4, 2, 3, 4, 2, 3,4)
x x x

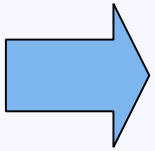
論理ベクトル

論理演算子

<, <=, >, >=, ==, !=, ! (否定), & (論理積), | (論理和)

論理演算子による結果は TRUE あるいは FALSE

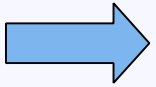
x ← 13 > 10  x = TRUE

x ← 1:10
y ← x > 5  y = (FALSE, FALSE, FALSE, FALSE, FALSE,
TRUE, TRUE, TRUE, TRUE, TRUE)

欠損値と非数

NA　欠損値を表す（現実のデータには欠損値がある）

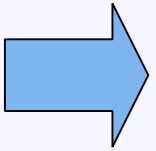
NA　に対する操作の結果は NA



とりあえず NA は考えなくてよいと思う

NaN　非数を表す（0/0 と Inf – Inf ）

```
x ← 1/0
y ← 0/0
z ← x – x
w← c(x,y)
xp ← is.nan(x)
yp ← is.nan(y)
zp ← is.nan(y)
wp ← is.nan(wp)
```



x = Inf, y = NaN, z = NaN, w = (Inf, NaN)
xp = FALSE, yp = TRUE, zp = TRUE,
wp = (FALSE, TRUE)

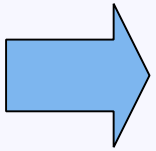
文字ベクトル

文字を要素とするベクトル、文字は二重引用符で囲む

x ← c(“shinnou”, “motegi”, “tanaka”, “toyokawa”)

paste 関数、、文字列を連結する

y ← paste(x,1:5,sep=“$”)



y = (“shinnou$1”, “motegi$2”, “tanaka$3”,
“toyokawa$4”, “shinnou$5”)

リサイクルされていることに注意

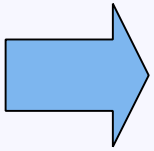
添え字ベクトル（1）

ベクトルの一部分を指定するベクトル。
[ ] で指定される。
４つのタイプが存在する。

注意
z = (8, 9, 10, 11,12)
ではない

（1）論理ベクトル

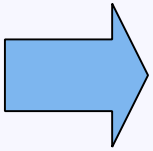
```
x ← 1:10
y ← x[x > 7]
z ← (x + 2)[x > 7]
```



y = (8, 9, 10)
z = (10, 11, 12)

（2）正の整数値ベクトル

```
x ← 2:10
y ← x[3:5]
```



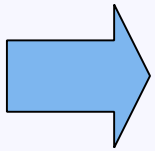
x = (2,3,4,5,6,7,8,9,10)
y = (4, 5, 6)

添え字ベクトル(2)

(3) 負の整数値ベクトル

正の逆、正で取り出されないものを取り出す

```
x ← 2:10
y ← x[-(3:5)]
```



```
x = (2,3,4,5,6,7,8,9,10)
y = (2, 3, 7, 8, 9, 10)
```

(4) 文字ベクトル

オブジェクトが名前属性を持つときに、その文字で示された名前をもつものを取り出す。先の学習になるので省略。

宿題

（1） [“X-1”, “X-2”, ..., “X-100”, “Y-1”, “Y-2”, ..., “Y-100”, “Z-1”, “Z-2”, ..., “Z-100”]
という300次元のベクトルを作成せよ。

（2） 100 から 200 までの整数値の総和を求めよ。

（3） 10000 ～ 100000 の整数の中で、13の倍数の数値はいくつあるかを
調べよ

ヒント：（2）、（3）はそのようなベクトルを作り、
sum や length を利用すればよい。

〆切：4月14日（月）17：30 まで