クレジット：
UTokyo Online Education 統計データ解析 Ⅱ 2018 小池祐太

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

統計データ解析 (II) 第 2 回

小池祐太

2017 年 4 月 19 日

基本的な使い方: 数の扱い

- R では実数及び複素数を取り扱うことができ, 指数表記にも対応している
- また, 無限大や不定な数など特殊なものを扱うこともできる
- 実行例 `numbers.r`

基本的な使い方: 変数への代入

- 文字列を変数名として, 数値等を保持することができる. また, 変数をそのまま計算に用いることもできる
- 文字列 `xxx` を変数名とする変数に `a` という値を代入するには, コンソール上で以下のコマンドを実行:

```
xxx <- a      または      xxx = a
```

- なお, R では, 変数や関数, および関数の実行結果等を総称して「オブジェクト」と呼ぶ
- 実行例 `variables.r`

基本的な使い方: 変数への代入

- 変数名は自由に決めて用いることが出来る (例: `x`, `y`, `abc` など)
- しかし, `sin`, `log`, `pi` など R の仕様として使われているものは, 用いることができないわけではないが混乱を招く元なので使わないほうがよい
- 参考 以下の文字は R 起動時からすでに特定の機能を与えられているので, 値を代入する際は注意が必要

  `c q t C D F I T`

  - それぞれの機能はヘルプを参照

基本的な使い方: 変数への代入

- これまでの作業で生成した変数に関する情報は, 右上のペインの “Environment” タブで確認できる
- また, これまでの作業でコンソール上で打ち込んだコマンドは, 右上のペインの “History” タブで確認できる
- 更に, コンソール上で上下キーを打つことで, 以前に実行したコマンドを再表示できる

パッケージのインストール

- R では, その機能を拡張するために多数のパッケージが用意されている
- 従って, 初期設定で関数が実装されていなくても, その関数を実装しているようなパッケージがすでに開発されていることが多い
- パッケージのインストールには以下の 2 通りの方法がある:
  - RStudio の機能を利用する方法
  - コンソールから行う方法

パッケージのインストール

- パッケージ **tseries** のインストール手順 (RStudio の機能を利用)
  1. 右下ペインの “Package” タブをクリック
  2. 左上あたりに “Install” という表示があるのでそれをクリック
  3. 新規ウィンドウが現れるので, “Package” のフォームにインストールしたいパッケージ名 (ここでは tseries) を入力し, 下部の “Install” をクリック
  4. パッケージがインストールされる

パッケージのインストール

- なお, インストールしようとしているパッケージがすでに手元にあり, 作業環境にロードされている場合, “Updating Loading Packages” というウインドウが現れるので, Yes を選択する. もし失敗した場合は再度同じウィンドウが出現するので, No を選択する. それでも失敗する場合は RStudio を一旦終了してやり直す

パッケージのインストール

- インストール済みのパッケージは右下ペインの “Package” タブに表示される
  - パッケージ名の左側のボックスをチェックすると, そのパッケージがロードされて, パッケージに含まれる関数などが利用可能になる
  - 逆に, チェックされているボックスを再度クリックすると, そのパッケージをアンロードできる
- 左上部の “Update” をクリックすると, インストール済みパッケージを最新版に更新できる

パッケージのインストール

- パッケージ **tseries** のインストール手順 (コンソールから)
  1. R のコンソール上で `install.packages("tseries")` を実行
  2. パッケージをダウンロードするためのサイト (CRAN のミラーサイト) を選ぶことを要求された場合は, 適当なものを選ぶ (Japan のミラーサイトがよい)
  3. パッケージがインストールされる
- インストールしたパッケージはコンソール上で `library(パッケージ名)` か `require(パッケージ名)` を実行することでロードされる (今の場合 `library(tseries)` または `require(tseries)` を実行)

データ構造

- R には, 以下のようなデータ構造が用意されている (代表的なもの):
  - ベクトル (vector)
  - 行列 (matrix)
  - リスト (list)
  - データフレーム (data frame)

データ構造: ベクトル

- ベクトルとは, スカラー値の集合
- R オブジェクトは基本的にはベクトルとして扱われる (スカラー値は長さ 1 のベクトルとして扱われる)
- スカラー値として扱われるものには, 実数や複素数以外に, 文字列 (“” で囲まれた文字. “x” など) や論理値 (TRUE, FALSE) などが含まれる
- 実行例 `scalar.r`

データ構造: ベクトル

- 要素 a, b, ... からなるベクトルは以下で生成できる:

```
c(a,b,...)
```

  - a, b, ... は数値や文字列が混同していてもよいが, 生成されたベクトルの各成分は同じデータ型に統一されることに注意
- ベクトル x の i 番目の要素は x[i] で取り出せる (注: ベクトルの添え字は 1 から始まる)
- x[c(1,3,4)] のように添え字もベクトルで指定すると, 指定された添え字に対応する要素からなるベクトルが取り出せる (この場合ベクトル c(x[1],x[3],x[4]))
- ベクトル x の長さは length(x) で取り出せる

データ構造: ベクトル

- 実数 x から y (x<y) まで 1 ずつ増加していく要素を持つベクトルは x:y で生成できる
  - x>y の場合は x から y まで 1 ずつ減少していく要素を持つベクトルとなる
- より一般に, 実数 x から y (x<y) まで a ずつ増加していく要素を持つベクトルは seq(x,y,by=a) で生成できる
- ベクトル x を n 回繰り返した要素をもつベクトルは rep(x,n) で生成できる
  - 従って, rep(x,n) の長さは length(x) × n となる
- ベクトル x の末尾にベクトル y をつなげたベクトルは c(x,y) で, x の要素を反転したベクトルは rev(x) で生成できる
- 実行例 vector.r

データ構造: 行列

- 行列 スカラー値を長方形状に並べたもの
  - 横の並びを**行**, 縦の並びを**列**と呼び, 行数 m, 列数 n の行列を $m \times n$ 行列と呼ぶ
  - 行数と列数のペアをその行列の**サイズ**と呼ぶ
  - 行数と列数が等しい行列を**正方行列**と呼ぶ. $n \times n$ 行列は n 次正方行列と呼ばれる
  - $m \times 1$ 行列, $1 \times n$ 行列はそれぞれ m **次元列ベクトル**, n **次元行ベクトル**と呼ばれる
  - 行列の各成分の行番号と列番号を入れ替えて得られる行列を**転置行列 (transposed matrix)** と呼ぶ (R では関数 `t()` で計算できる)
- ベクトルの場合と同様, 各成分は同じデータ型に統一される

データ構造: 行列

- すべての要素が a である $m \times n$ 行列は

  `matrix(a,m,n)`

  で生成できる
- より一般に, 長さ mn のベクトル
  $x = (x_{11}, \ldots, x_{m1}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn})$ に対して,

  `matrix(x,m,n)`

  で $m \times n$ 行列 $(x_{ij})_{1 \leq i \leq m, 1 \leq n}$ が生成できる
  - `as.vector` を適用することでベクトル化できる:

    ```
    x = as.vector(matrix(x,m,n))
    ```

データ構造: 行列

- 行列 `X` のサイズは `dim(X)` で取り出せる (長さ 2 のベクトル)
- 逆に, ベクトルに dim 属性を指定してやることで行列に変換できる
  - 例えば, 前頁のベクトル `x` に対して

    ```
    dim(x) <- c(m,n)
    ```

    を実行すれば, `x` は行列 `matrix(x,m,n)` に変換される
- 行列 `X` の行数は `nrow(X)` で, 列数は `ncol(X)` で取り出せる
- 行列 `X` の (i, j) 成分は `X[i,j]` で取り出せる
  - ベクトルの場合と同様に添え字をベクトルで指定することで, 部分行列の取り出しも可能
  - `X[i,]`, `X[,j]` でそれぞれ `X` の第 `i` 行, 第 `j` 列が取り出せる

データ構造: 行列

- 長さが等しい複数のベクトル `x`, `y`, ... を列もしくは行ベクトルとする行列を作成することもできる
  - 列ベクトルの場合 ··· `cbind(x, y, ...)`
  - 行ベクトルの場合 ··· `rbind(x, y, ...)`
- `cbind`/`rbind` は行数/列数が等しい複数の行列を横/縦に結合するのにも使える
- 補足 行列の高次元版のデータ構造として, 配列 (array) が用意されている
- 実行例 `matrix.r`

データ構造: リスト

- リスト
  - 異なる構造のデータをまとめて 1 つのオブジェクトとして扱えるようにしたもの
  - リストの各要素はデータ型・クラスともにバラバラであってもよい
- R のオブジェクト x, y, ... を要素とするリストは `list(x,y,...)` で生成される. リスト L の i 番目のオブジェクトには `L[[i]]` でアクセスできる.

データ構造: リスト

- リストは各成分に名前をつけることができる (実はベクトルも)
  - 方法 1: 作成時に `list(name1 = x, name2 = y, ...)` のように書くと, 各成分に `name.1`, `name.2`, ... といった名前がつく
  - 方法 2: すでに作成してある長さ `n` のリスト `L` の各成分に名前をつける (もしくは名前を変更する) には,

    ```
    names(L) <- c(name.1, name.2, ..., name.n)
    ```

    のようにする
- リスト `L` の名前 `name` の成分は, `L$name` もしくは `L[["name"]]` で取り出せる
- 実行例 `list.r`

データ構造: データフレーム

- データフレーム
  - 行列風リスト
  - リストにおいて, 各成分が長さが等しいベクトルであるようなもの (各成分を列と考えると行列のように表現できる)
  - 特に, 列ごとにデータ型はバラバラでも良い
- イメージ
  - 複数の個体について, いくつかの属性を集計したデータ (例えばある小学校の 1 年生の身長, 体重, 性別, 血液型, ... を集計したデータ)
  - リストの各成分はある属性に関する観測データに対応
  - 個体数は集計項目に関わらず変化しないが, 集計項目によっては定量的データ・定性的データの違いが出てくるので, データ型は変わりうる

データ構造: データフレーム

- 長さが等しいベクトル `x`, `y`, ... を成分とするデータフレームは, `data.frame(x,y,...)` で生成される
  - データフレームはリストなのでリストと同様にして各変数にアクセスできる
  - 他方, データフレームは行数がベクトルの長さ (個体数), 列数が変数の個数 (観測項目の数) の行列と同様にアクセスできる
  - 引数 `row.names` を指定することで行名もつけられる
- 実データは上のような表形式のデータであることが多いので, 実データを R に読み込む際に役に立つデータ構造
- 実行例 `data.frame.r`

ベクトルの計算

- 以下ではベクトルを太字で (対応する R オブジェクトはタイプライタ体), その要素は下付き添字で表現する
- 例えば k 次元ベクトルは

$$\boldsymbol{a} = (a_1, a_2, \ldots, a_k) \tag{1}$$

のように表す

ベクトルの計算: 和

- 同じ長さのベクトルの和および差

$$\boldsymbol{a} \pm \boldsymbol{b} = (a_1 \pm b_1, a_2 \pm b_2, \ldots, a_k \pm b_k) \tag{2}$$

  は, 数値の和と差のように扱うことができる
- すなわち, a, b が同じ長さのベクトルであれば,

```
a+b
```

  によって (2) を計算できる
- 実行例 `vector-sum2.r`

ベクトルの計算: 積

- 数値の場合と同様に, 同じ長さの 2 つのベクトル a と b の積 a*b が定義されているが, これは成分ごとの積 (Hadamard 積)

$$\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, a_2 b_2, \ldots, a_k b_k) \tag{3}$$

  を計算する
- 一方, ベクトルに対する積は, 内積

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{k} a_i b_i \tag{4}$$

  の方がポピュラーであり, こちらは a%*%b で計算できる
- 実行例 `vector-prod2.r`

ベクトルの計算: 初等関数の適用

- ベクトルに初等関数 (sin, exp, . . . など) を適用すると, 成分ごとに計算した結果が返される
- 例えば, ベクトル **a** に関数 sin を適用した結果 `sin(a)` は

$$(\sin(a_1), \ldots, \sin(a_k))$$

  となる
- 実行例 `vector-fun.r`

行列とその演算

- 多変量解析で現れる統計量の計算を見通しよく行うためには, 行列に対する種々の演算を利用するのが便利である
- 以下では行列を大文字で (対応する R オブジェクトはタイプライタ体), その要素は下付き添字で表現する. 例えば $m \times n$ 行列は

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{5}$$

  のように表す
- また, 行列 A の ij 成分を指す場合には $(A)_{ij}$ のように書くこともある
- 以下では特に断らない限り, 実数を成分とする行列のみを扱う

行列とその演算: 和

- 同じ大きさの行列の和および差

$$(A \pm B)_{ij} = a_{ij} \pm b_{ij} \tag{6}$$

  は, ベクトルと同じように記述することができる
- すなわち, `A`, `B` が同じサイズの行列であれば,

```
A+B
```

  によって (6) を計算できる
- 実行例 `matrix-sum2.r`

行列とその演算: 積

- ベクトルの場合と同様に, サイズが同じ 2 つの行列 `A` と `B` に対して, `A*B` は成分ごとの積 (Hadamard 積)

$$(A \circ B)_{ij} = a_{ij} b_{ij} \tag{7}$$

  を計算する
- 一方で, A が $n \times m$ 行列, B が $m \times l$ 行列のとき, 通常の意味での行列に対する積 AB

$$(AB)_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \tag{8}$$

  が定義されるが (AB は $n \times l$ 行列), こちらは `A%*%B` で計算できる
- 実行例 `matrix-prod2.r`

行列とその演算: 初等関数の適用

- ベクトルの場合と同様, 行列に初等関数 (sin, exp, . . . など) を適用すると, 成分ごとに計算した結果が返される
- 例えば, 行列 A に関数 sin を適用した結果 `sin(A)` は, (i, j) 成分が

$$\sin(a_{ij})$$

で与えられる, 行列 A と同じサイズの行列となる
- 実行例 `matrix-fun.r`

行列とその演算: 行列式とトレース

- 連立 1 次方程式

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

は, n 次正方行列 A, n 次元列ベクトル $\boldsymbol{x}$ および $\boldsymbol{b}$ をそれぞれ

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

で定めると, 行列方程式

$$A\boldsymbol{x} = \boldsymbol{b} \tag{9}$$

の形で書くことができる

行列とその演算: 行列式とトレース

- 方程式 (9) の解の存在や一意性を議論する上で, 行列 A の**行列式 (determinant)** と呼ばれる量が重要な役割を果たす
- 行列 A の行列式は記号 $\det A$ で表される
- 一般の行列に対する行列式の定義は複雑なためここでは割愛するが, A が 2 次正方行列の場合は単純であり,

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

  となる
- R においては, 行列式は関数 `det()` を用いて計算することができる

行列とその演算: 行列式とトレース

- 正方行列の**トレース (trace)** は対角成分 (対角線上の成分) の総和として定義される
- 正方行列 A のトレースは記号 $\mathrm{tr}\, A$ で表されることが多い:

$$\mathrm{tr}\, A = \sum_{i=1}^{n} a_{ii}.$$

- R においては, トレースを計算するための専用の関数は用意されていないが, 対角成分を取り出す関数 `diag()` とベクトルの成分の総和を計算する関数 `sum()` を組み合わせて

```
sum(diag(A))
```

のように計算できる

- 実行例 `matrix-det3.r`

行列とその演算: 逆行列

- 対角成分以外の成分がすべて 0 であるような正方行列を**対角行列**と呼ぶ
- 特に, 対角成分がすべて 1 であるような対角行列を**単位行列**と呼ぶ
- 以下 n 次単位行列を記号 E_n で表す
- R では, n 次単位行列は

  `diag(n)`

  で生成することができる

行列とその演算: 逆行列

- n 次正方行列 A に対して, n 次正方行列 B が

$$AB = BA = E_n$$

を満たすとき, B を A の**逆行列 (inverse matrix)** と呼び, 記号 A^{-1} で表す (逆行列は存在すれば一意的に定まる)
  - 逆行列をもつような正方行列は**正則 (regular)** もしくは**非特異 (non-singular)** であるという
  - 逆行列は常に存在するとは限らない. 正方行列 A が正則である (つまり, 逆行列をもつ) ための必要十分条件は, A の行列式 $\det A$ が 0 でないことであるという事実が知られている
- 正則行列の逆行列を求めるには関数 `solve()` を用いる (正則でない行列に適用するとエラーとなる)

行列とその演算: 一般化逆行列

- 一般に A が (正則とも正方とも限らない) 行列の場合でも, 逆行列に類する概念として**一般化逆行列**を考えることができ, データ解析においてしばしば必要となる
- A の一般化逆行列とは,

$$AA^{\dagger}A = A \tag{10}$$

を満たす行列 $A^{\dagger}$ のことである

行列とその演算: 一般化逆行列

- 一般に A の一般化逆行列は一意には定まらないが, 以下の条件を満たす一般化逆行列 A^+ は一意的に定まることが知られている

$$AA^+A = A \tag{11}$$
$$A^+AA^+ = A^+ \tag{12}$$
$$(AA^+)^\top = AA^+ \qquad (* \text{ は転置行列の意}) \tag{13}$$
$$(A^+A)^\top = A^+A \tag{14}$$

- A^+ は A の **Moore-Penrose の一般化逆行列**と呼ばれている
- Moore-Penrose の一般化逆行列は, **MASS** パッケージの関数 `ginv` によって計算できる
- 実行例 `matrix-inv3.r`

ベクトルと行列の計算: 積

- R 言語においては，列ベクトル・行ベクトルという区別はなく，単一のベクトルという概念で扱われている
- このため, 行列とベクトルの積においては, 行列のどちらからベクトルを掛けるかによって自動的に列ベクトルか行ベクトルか適切な方で扱われる
- ただし, ベクトルも行列の一種であるから, 計算結果は行列として表現されることに注意する
- 実行例 `linear-calc2.r`

ベクトルと行列の計算: 連立方程式

- A が n 次正則行列, $\mathbf{b}$ が n 次元列ベクトルのとき, $\mathbf{x}$ に関する連立 1 次方程式

$$A\mathbf{x} = \mathbf{b}$$

  は解 $\mathbf{x} = A^{-1}\mathbf{b}$ をもつが, この解は関数 `solve` によって計算できる
- 書式

```
solve(A, b)
```

- なお, $\mathbf{b}$ はベクトルのかわりに行数 n の行列でも可
- 実行例 `linear-eq2.r`

固有値と固有ベクトル

- 一般に n 次正方行列 A に対して, 複素数 λ と零ベクトルでない n 次元ベクトル $\mathbf{x}$ が

$$A\mathbf{x} = \lambda\mathbf{x}$$

  を満たすとき, λ を A の固有値, $\mathbf{x}$ を λ に対する固有ベクトルと呼ぶ
- 正方行列 `A` の固有値と固有ベクトルは `eigen(A)` で計算できる. 結果は固有値と固有ベクトルを列ベクトルとする行列からなるリストで返される

固有値と固有ベクトル

- A の転置行列 $A^\top$ が A 自身に一致する場合 ($A^\top = A$), A は**対称**であるという
- A が対称ならば, A の固有値はすべて実数であり, かつ A の固有ベクトルを n 個並べて得られる正則行列 V で, $V^{-1}AV$ が対角行列となるようなものがとれることが知られている (この場合 $V^{-1} = V^\top$ となるようにとれる)
- この操作を A の**対角化 (diagonarization)** と呼ぶ
- 実行例 `eigen.r`

関数定義

- 多くの計算機言語と同様, R でもユーザー独自の自作関数を定義できる
- 自作関数の定義には関数 `function` を利用する
- 例 半径 r から球の体積と表面積を求める関数 `myfunc`

```
myfunc <- function(r){
    V <- (4/3) * pi * r^3 # 球の体積
    S <- 4 * pi * r^2 # 球の表面積
    out <- c(V,S)
    names(out) <- c("volume", "area")
    # 返り値に名前をつける
    return(out)
}

myfunc(1) # 実行
```

制御文

- 一般に最適化や数値計算などを行うためには, 条件分岐や繰り返しを行うための仕組みが必要となる
- R 言語を含む多くの計算機言語では `if` (条件分岐), `for`・`while` (繰り返し) を用いた構文が用意されているが, これを制御文と言う
- 実行例 `control2.r`

制御文

- `if` 文: 書式

$$\texttt{if(条件 A)\{ プログラム X\}}$$

  - 条件 A が満たされる場合, プログラム X を実行する
  - 「条件 A が満たされない場合, 代わりにプログラム Y を実行する」としたければ, 以下のように書く:

$$\texttt{if(条件 A)\{ プログラム X\}else\{ プログラム Y\}}$$

- `for` 文: 書式

$$\texttt{for(i in M)\{ プログラム X\}}$$

  - `M` をベクトルとし, 変数 `i` が `M` の要素となる値すべてを動きながらプログラム X を繰り返し実行する
  - プログラム X は通常変数 `i` によって実行内容が変わる

制御文

- `while` 文: 書式

$$\texttt{while(条件 A)\{ プログラム X\}}$$

  - 条件 A が満たされる限り, プログラム X を繰り返し実行する
  - プログラム X は通常実行するたびに実行内容が変わり, いつか条件 A が満たされなくなるように書く