

# R の導入と使い方

宮田 敏

2024 年 4 月 10 日

本稿では、フリーの統計解析ソフトウェア R の導入と使い方の紹介を行う。ただし、R の使い方については本書を読むのに必要な最小限の説明にとどめる。この付録の内容では R を使いこなすには全く不足するので、全般的な R の詳細については他書（船尾 [1] など）を参照されたい。

ソフトウェア R は統計計算とグラフィックスのための言語・環境であり、R と付属文書は GNU general public licence (<http://www.gnu.org/licenses/gpl.html>) の下に自由に配布されている。R は Windows, Mac OS, Linux などの OS 上にインストール可能であるが、本書は Windows の利用を前提とする。他の OS を使用する読者は、例えば RjpWiki (<http://www.okada.jp/RWiki/>) の該当部分などを参照し適宜読み替えていただきたい。本書執筆時点での、R のバージョンは R 4.3.3 である（2024 年 4 月 10 日現在）。R のバージョンは頻繁に更新されるが、さほど最新版にこだわる必要はない。

## 1 R のインストール (Windows)

R のソースコードおよびそのバイナリは Comprehensive R Archive Network (CRAN) のウェブサイト (<http://www.r-project.org/>) あるいはそのミラーサイトから入手することが出来る。R のセットアッププログラムを入手するため、インターネットに接続した PC 上でブラウザを起動し、以下の URL にアクセスする。

<https://cloud.r-project.org/>

図 1 が表示されるので、**Download R for Windows** を選択する。Mac OS, Linux 等を用いる場合は、それぞれ該当するものをダウンロードする。次に、図 2 が表示されるので、**Subdirectionaries:** の下の **base** を選択する。

Windows 用 R セットアップファイルダウンロード画面 (図 3) で、**Download R 4.0.2**

図 1

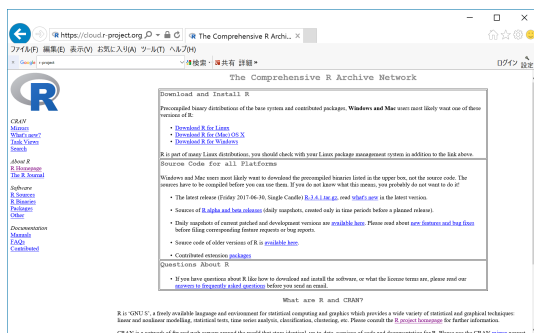
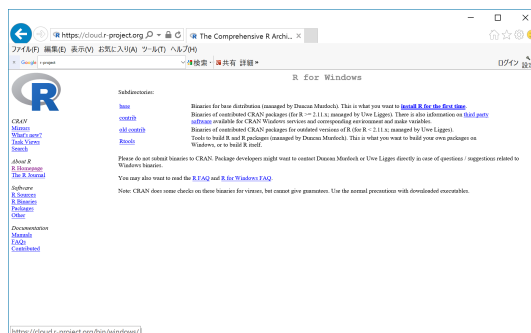


図 2



for Windows をクリックする。セットアップファイル (R-4.3.3-win.exe) を選択し適当なフォルダ (例えばデスクトップに) にダウンロードする。(図 4. 図では R-3.4.2-win.exe になっているが気にしない)

図 3

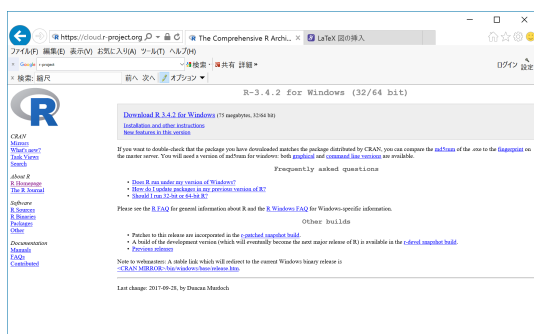
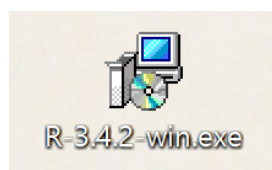


図 4



セットアップファイル (R-4.3.3-win.exe) をダブルクリックすると、「ユーザーアカウント制御」ダイアログボックスが表示されるので、「はい」を選択する。「セットアップに使用する言語の選択」ダイアログボックスが開くので、「日本語」を選択し「OK」を押す。「R for Windows 4.3.3 セットアップ」ダイアログボックスと「GNU GENERAL PUBLIC LICENSE」に関する説明が表示されるので、どちらも「次へ」を押す。次の「インストール先の指定」「コンポーネントの選択」ダイアログボックスも、特に理由がなければそのまま「次へ」「次へ」を押す。「起動時オプション」の選択、「プログラムグループの指定」も、そのまま「次へ」を押す。

「追加タスクの選択」(図 5) では、必要な追加タスクを選択する。デフォルトでは「クイック起動アイコンを作成する」はチェックされていないが、これも選択しておくとも便利

である。「次へ」を押すと、R のインストールが開始される。インストールが終了したら、「完了」ボタンを押してインストールを完了する。デスクトップには R のアイコン（図 6）が出来ているはずである。

図 5

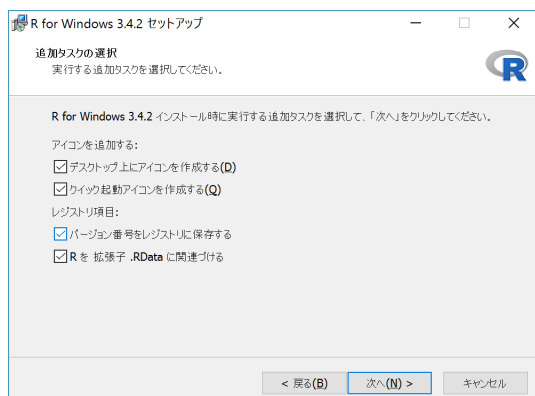


図 6



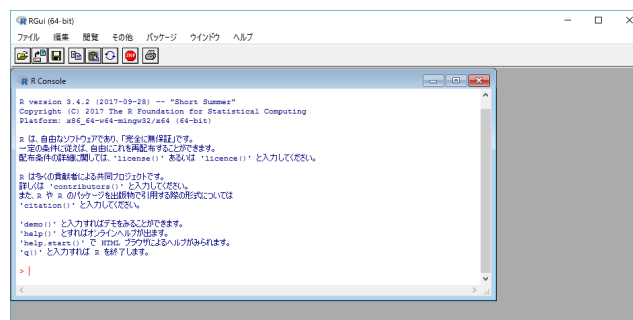
## 2 R の起動と終了

さて、R がインストールされたら、早速 R を使ってみよう。R を起動するには、以下の方法がある。

- デスクトップ上の R のアイコンを、ダブルクリックする。
- 「スタート」ボタンから、「すべてのプログラム」→「R」→「R 4.3.3」をクリックする。

R を起動すると、起動直後の画面は図 7 のようになる。初期画面の中には「R Console」

図 7 R の初期画面



というウィンドウがある。この R Console ウィンドウの中にコマンドを入力すると、同じ画面に計算結果が出力される。その意味で、R は「対話型プログラム」と呼ばれる。例えば  $\log(10)$  を計算すれば、以下の結果を得る。

```
> log(10)
[1] 2.302585
```

2 行目の先頭にある “[1]” は、出力結果の要素の数が 1 つ、という意味である。R では、四則演算や上記の対数関数  $\log()$  の他、指数関数  $\exp()$ 、三角関数  $\sin()$ 、 $\cos()$ 、 $\tan()$ 、逆三角関数  $\asin()$ 、 $\acos()$ 、 $\atan()$  などの初等関数の値を求めるコマンドがある。さて、R を使い始めたばかりであるが、ここでいったん R を終了してみよう。R を終了するには、以下の方法がある。

- 「ファイル」メニューから「終了」を選択する。
- R の画面の右上隅の「×」印を押す。
- R Console ウィンドウの中で、終了コマンド  $q()$  を実行する。

```
> q()
```

R 終了時には「質問」という小さなウィンドウが開き、「作業スペースを保存しますか？」と聞かれる。「はい (Y)」を選択すればそれまでの作業を保存し次回再利用することが出来る。しかし、「いいえ (N)」を選択し何も保存しないことを、強くお勧めする。いたずらに作業を保存しても、次回までに保存した内容を忘れてしまい要らぬ混乱を招くのが関の山である。もし R 内部に保存されたものが分からなくなったときは、「その他」メニューから「その他」→「すべてのオブジェクトの消去」と辿ると「質問」ダイアログボックスで「本気ですか？」と聞かれるので、「はい」を押して保存されたものをすべて消去してしまう。R の中は、いつもきれいに掃除しておこう。

## 3 R の操作

### 3.1 オブジェクトと付値

それでは、ふたたび R を起動しよう。R において扱われる様々な変数、関数、データなどの「もの」は、すべて**オブジェクト (object)** と呼ばれる。R において最も基本的なオブジェクトは、**ベクトル (vector)** である。R においてベクトルは、R のコマンド  $c()$  によって定義される。例えば、以下の構文を R Console ウィンドウに入力することで、要

素の数が 5 の実数値ベクトルを定義することが出来る．

```
> x <- c(3, 4, 10, 6, 5)
> x
[1] 3 4 10 6 5
```

ただし、1 行目の “<-”（不等号 “<” とハイフン “-” を続けて入力する）は左向き矢印を意味し、「右辺」（この場合は `c(3, 4, 10, 6, 5)`）の実行結果を「左辺」のオブジェクト（この場合は `x`）に代入し保存するための記号である．このようにあるオブジェクトの値を他のオブジェクトに代入することを、**付値 (assign)** する、という．上の例の場合、`c()` コマンドの実行結果が付値されたためオブジェクト `x` の値もまた、要素数 5 の実数値ベクトル (3, 4, 10, 6, 5) となったことを示している．

## 3.2 データの型

ベクトルの要素となるデータの型（属性）には、上に挙げたオブジェクト `x` のような**数値型 (numeric)** の他、**文字列型 (character)**、**論理型 (logical)**、**因子型 (factor)** などがある．文字列型ベクトルは、引用符 (“ ” あるいは ‘ ’) で囲まれた文字列を要素として持つベクトルである．例えば、以下の文字列型ベクトルは長さ 3 のベクトルで、最初の要素は 4 文字のアルファベットからなる “jack” という文字列である．

```
> y <- c("jack", "eric", "ginger")
> y
[1] "jack" "eric" "ginger"
```

論理型ベクトルは、**TRUE** もしくは **FALSE** の論理値を要素に持つベクトルである．上に定義したオブジェクト `x` を用いて、以下の論理式を実行してみよう．

```
> (x > 4.5)
[1] FALSE FALSE TRUE TRUE TRUE
```

`x` の要素は 3, 4, 10, 6, 5 であるが、その一つ一つに対して「4.5 より大きい ( $x > 4.5$ )」という命題が評価され、例えば `x` の最初の要素である 3 は 4.5 より大きくはないため **FALSE (偽)** という論理値が、三番目の要素である 10 は 4.5 より大きいため **TRUE (真)** という論理値が返されている．

もう一つのデータの型である因子型ベクトルは、同じ長さを持つ別のベクトルの要素の

グループ化を行うデータ型であり、`factor()` コマンドによって定義される。以下の例を考えよう。

```
> z <- c("jack", "jack", "ginger", "eric", "eric")
> z
[1] "jack"  "jack"  "ginger" "eric"  "eric"
> z.f <- factor(z)
> z.f
[1] jack   jack   ginger eric    eric
Levels: eric ginger jack
```

最初の行では、文字列型ベクトル `z` を定義している。`z` を表示すると、`z` の要素が "jack" のように文字列であることを示す引用符 " 付きで示されることが分かる。一方、`z` に `factor()` コマンドを作用させた `z.f` は因子型に**型変換**され、因子型ベクトルになっている。`z.f` を表示すると、`z.f` が文字型でない証拠に `z.f` の要素には引用符 (" ") がついていない。また `z.f` は要素のグループ分けとして `eric`, `ginger`, `jack` の三つの**レベル (Level)** を持つことが分かる。レベルの名前 (**ラベル (Label)**) を変えるには、`labels` オプションで指定する。例えば、`eric`, `ginger`, `jack` を `ERIC`, `GINGER`, `JACK` に変えるには以下のようにする。

```
> z.f <- factor(z, labels=c("ERIC", "GINGER", "JACK"))
> z.f
[1] JACK   JACK   GINGER ERIC    ERIC
Levels: ERIC GINGER JACK
```

因子型のレベルは、数字の大小順あるいはアルファベット順に順序を持つ。もしそれ以外の順序を指定する際は、以下のように `levels` オプションで指定する。

```
> factor(z, levels=c("jack", "eric", "ginger"))
[1] jack   jack   ginger eric    eric
Levels: jack eric ginger
```

このようにすると、`z.f` のレベルがアルファベット順ではなく、指定されたとおり `jack`, `eric`, `ginger` となっていることが分かる。

### 3.3 行列, リスト, データフレーム

ベクトルは、行列の形にまとめることも出来る。R で行列を定義するコマンドは、`matrix()`、`cbind()`、`rbind()` などがある。 `matrix()` は入力されたベクトルを行列の形に変形する。以下にその例を示す。

```
> mtx <- matrix(1:8, nrow=2, ncol=4)
> mtx
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> mtx2 <- matrix(1:8, nrow=2, ncol=4, byrow=TRUE)
> mtx2
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

ここで、`1:8` の “:” は `1, 2, ..., 8` のような数列を生成する関数である。また、`matrix()` コマンドの中の `nrow`, `ncol` はそれぞれ行数, 列数を指定するオプションである。上の例にあるとおり、`matrix()` コマンドは入力されたベクトルの要素を列優先で並べるが、`byrow=TRUE` オプションを与えれば行優先で並べられる。これに対して、`cbind()`、`rbind()` は column bind, row bind の意であり、同じ長さの複数のベクトルをそれぞれ列ごとあるいは行ごとに行列の形にまとめる。

```
> x1 <- 1:3
> x2 <- 5:7
> mtx3 <- cbind(x1, x2)
> mtx3
      x1 x2
[1,]  1  5
[2,]  2  6
[3,]  3  7
> mtx4 <- rbind(x1, x2)
```

```
> mtx4
      [,1] [,2] [,3]
x1      1   2   3
x2      5   6   7
```

これまで述べてきたベクトル、行列は、その要素がすべて同じデータの型をしている必要があった。これに対して、**リスト (list)** は異なる型と異なる長さを持ったベクトルや行列などのオブジェクトを一纏めにしたものである。リスト型オブジェクトは `list()` コマンドによって定義される。これまで作ったオブジェクトを使って、以下の例を考えよう。

```
> L <- list(x, y, z, z.f)
> L
[[1]]
[1] 3 4 10 6 5

[[2]]
[1] "jack" "eric" "ginger"

[[3]]
[1] "jack" "jack" "ginger" "eric" "eric"

[[4]]
[1] jack jack ginger eric eric
Levels: jack eric ginger
```

ここでリスト `L` は、最初の要素が長さ 5 の数値型ベクトル、2 番目の要素が長さ 3 の文字列型ベクトル、等、長さも属性も異なる要素を持ったリストになっている。リスト型オブジェクトの特別な場合として、**データフレーム (data frame)** と呼ばれるものがある。データフレームとは、長さの等しいベクトルを纏めたリストであり、個々のベクトルの型は異なってもかまわない。データフレームは、`data.frame()` コマンドによって定義される。

```
> Data <- data.frame(id = 1:5,
group = c("A", "A", "B", "B", "A"),
x = c(5,2,3,8,1))
```



```
> Data
  id group x
1  1     A 5
2  2     A 2
3  3     B 3
4  4     B 8
5  5     A 1
```

データフレームの形は行列と同様であるが、各列ごとに「変数名」に当たるベクトル名 (id, group, x) が付いている。また、上の例では Data オブジェクトの二番目の要素 group は `group = c("A", "A", "B", "B", "A")` のように文字型として定義されているが、`data.frame()` コマンドでデータフレームの要素として定義された後は因子型に型変換されている (引用符 `"` がない) ことに注意する。

## 4 要素の取出し

ここまでに導入したベクトル、行列は、数字、文字などのいくつかの要素をまとめたものである。また、リスト、データフレームはいくつかのベクトル、行列をまとめたものであった。データを操作する際これらのオブジェクトの一部を取り出すには、その要素を指定 (indexing) することで行われる。ベクトル、行列の要素の指定は、オブジェクト名の後ろに四角括弧 `[...]` を付け、1) 要素の場所を数字で指定する、あるいは 2) 論理値により、取り出す要素を指定する。例えば上の例でオブジェクト `x` は、長さ 5 の数値型ベクトルである。

```
> x
[1]  3  4 10  6  5
```

数字で要素の場所を指定する、例えば `x` の 3 番目と 5 番目の要素を取り出すには、`x` の後に付けた四角括弧 `[...]` の中に取り出す要素の位置を示す数値ベクトルを入れて、以下のようにする。

```
> x[c(3, 5)]
[1] 10  5
```

行列の場合、第  $i$  行を取り出すなら `mtx[i,]`、第  $j$  列を取り出すなら `mtx[,j]`、 $(i, j)$  要

素を取り出すなら、`mtx[i, j]` のように指定する．複数の行あるいは列を指定することも出来る．

```
> mtx
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> mtx[1,]
[1] 1 3 5 7
> mtx[,3]
[1] 5 6
> mtx[2,4]
[1] 8
> mtx[(1:2), c(2,4)]
      [,1] [,2]
[1,]    3    7
[2,]    4    8
```

ベクトルから論理値を用いて要素を取り出す場合は、ベクトルと同じ長さの論理ベクトルを用意する．論理ベクトルの要素が TRUE（真）である場所に対応する要素が取り出される．

```
> x > 4.5
[1] FALSE FALSE  TRUE  TRUE  TRUE
> x[x > 4.5]
[1] 10  6  5
```

一行目の `x > 4.5` は `x` と同じ長さの論理ベクトルであるが、その下では  $(x > 4.5)$  なる条件が満たされた要素のみが（=TRUE に対応する要素のみが）返されていることが分かる．行列の場合も同様に、論理ベクトルを使って取り出す行および列を指定することができる．

リスト、データフレームの場合、その要素はベクトル、行列などのオブジェクトである．リスト、データフレームの要素には二重四角括弧 `[[...]]` を使ってアクセスする．前に定義したリスト `L` の 1 番目の要素であれば `L[[1]]`、1 番目の要素のベクトルの 2 番目の要素であれば `L[[1]][2]` で取り出せる．

```
> L[[1]]
[1] 3 4 10 6 5
> L[[1]][2]
[1] 4
```

リスト、データフレームの要素が名前を持っていれば、`[...]` の中で名前を指定する、`$`の後ろに名前を続ける、の2通りの方法で要素の名前を指定できる。以下の4通りの方法は、データフレーム `Data` の3つ目の要素であるベクトル `x` を指定している。

```
> Data[[3]]
[1] 5 2 3 8 1
> Data[["x"]]
[1] 5 2 3 8 1
> Data$x
[1] 5 2 3 8 1
> Data[,3]
[1] 5 2 3 8 1
```

## 5 外部データの入出力

実際の解析において用いられるデータの多くは、Excel などの表計算ソフトのワークシートとして保存されている。本節では、外部データの形で保存されたデータを R に入力する、あるいは R 内部で作られたデータを外部に出力する方法を解説する。

### 5.1 データの準備

本節では一度 R を離れ、まず Excel などを用いて元のデータを作成、保存するところから始める。どのような解析を行うにせよ、データを作成する際は必ずやらなければならないこと、逆に絶対にやってはいけないことがある。最低限やるべきことをルーチンワークとして行うだけで、データ解析のミスを大幅に減らすことができる。

1. **データの形は長方形:** データを入力する際は、第一行目に変数名を記入する。多くのソフトウェアは日本語入力に対応しているが、**全角文字は避ける**方が無難である。第二行目以降にデータを記入するが、元データにはグラフや解析の結果を張り

付けたりはしない。そうすると、元データの形は、以下のような「長方形」になるはずである。

ID	sex	age	height	bodyweight
1	M	64	173	75.4
2	M	69	164	72
3	M	78	155.2	47.2
4	M	83	159.1	60
5	F	73	147.6	40.5

2. **元データは絶対に改変しない:** データを解析する際、変数を変換したり新しい変数を定義したりする必要があることがある。このとき、元データを改変してデータを上書きしたり、新しい変数を付け加えたりしてはいけない。データを改変したときは、必ず**新しいファイル名で保存**する。元データを改変した場合、解析を進めるうちに元データが何であったかわからなくなることがある。元データが分からなくなれば、**意図せざるデータの捏造**まであと一歩である。
3. **個人情報に記載しない:** ヒトのデータを扱う場合、個人情報の保護の重要性は改めて述べるまでもないが、残念ながらいまだに氏名やカルテ番号など個人を特定できる情報を記載したままでデータをやり取りする例がみられる。個人情報はデータ解析の立場からは何の意味もないが、万が一外部に流出した場合、データ元の方のプライバシーと研究そのものに重大な被害を与えることになる。解析データの**個人情報**は削除する、を徹底する必要がある。

## 5.2 R への外部データの読み込み

前節のようにしてデータを作成したら、それを Excel で保存する。本書では、サンプルデータとして `appA_Rintro_Data.xlsx` を用意した。これには  $x_1$ ,  $x_2$  の二つの変数、各 100 個のデータが含まれている。

1. **タブ区切りテキストファイル形式の保存:** R は Excel ファイルを含め様々なファイル形式のデータを読み込むことができるが、もっとも簡単なのはテキストファイル形式である。テキストファイルにも「タブ区切りテキスト」「CSV (カンマ区切り)」などがあるが、ここでは「タブ区切りテキスト」形式に絞って説明する。

Excel でファイルをタブ区切りテキスト形式で保存するには、ファイルメニューから「ファイル」→「名前を付けて保存」を選択し、「ファイルの種類 (T)」から「テキスト (タブ区切り) (\*.txt)」を指定して「保存 (S)」ボタンを押す (図 8)。

図 8

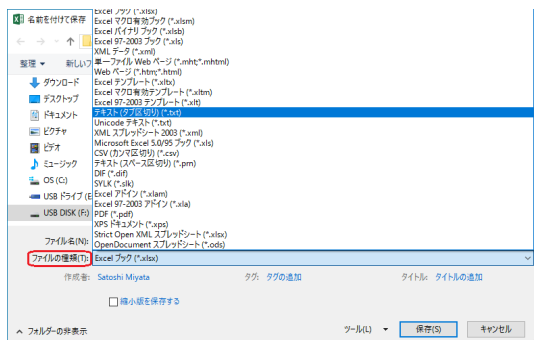


図 9



2. 作業ディレクトリ (working directory) の指定: R に外部データを読み込むため、R を起動する。R では外部データを保存したり、逆にデータやグラフなどの出力先となるディレクトリを作業ディレクトリ (working directory) と呼ぶ。R 起動時の作業ディレクトリは、`getwd()` コマンドで確認できる。

```
> getwd()
```

```
[1] "C:/Users/***/*/Documents"
```

R で作業ディレクトリを指定するには、以下の二通りの方法がある。

- R Console ウィンドウをアクティブにする。(R Console ウィンドウの上で、マウスをクリックする) 「ファイル」メニューから「ファイル」→「ディレクトリの変更...」を選択する。「フォルダーの選択」ダイアログボックスが表示されるので、作業ディレクトリを探して指定する。
- R Console ウィンドウから、`setwd()` コマンドで指定する。`setwd()` コマンドには、作業ディレクトリの絶対パスを入力する。絶対パスは、エクスプローラーの上部にあるアドレスバーをクリックすれば表示される (図 9)。Windows ではパス名の中のフォルダーの区切りはバックスラッシュ “\” であるが、R ではスラッシュ “/” であることに注意する。

```
> setwd("C:/Rdataset")
```

```
> getwd()
```

```
[1] "C:/Rdataset"
```

ここでは例として、C ドライブの直下に “Rdataset” フォルダを作り、これを作業ディレクトリとする。フォルダを作る際は、Windows エクスプローラーを起動し C ドライブの直下に移動したうえで、ファイルメニューから「ファイル」→「新規作成」→「フォルダ」を選択する。フォルダ名（たとえば Rdataset）を指定すれば完成である。今作成した **C:¥Rdataset** フォルダに、上で作成したタブ区切りテキストファイル `appA_Rintro_Data.txt` を保存する。

3. `read.table()` コマンドによる外部データの読み込み: 作業フォルダに保存したタブ区切りテキストファイルを R に読み込むコマンドは、`read.table()` コマンドである。`read.table()` の引数は引用符"で囲んだファイル名であり、いくつかのオプションが付く。

```
> Data <- read.table("appA_Rintro_Data.txt", header=TRUE, sep="\t")
```

上では、右辺で `read.table()` コマンドを実行し、読み込まれたデータを左辺の “Data” オブジェクトに保存している。`read.table()` コマンドの “header” オプションは、元データの 1 行目に変数名がある（ヘッダーがある）場合には `header=TRUE`、変数名がないときは `header=FALSE` となる。（“TRUE”, “FALSE” は、それぞれ “T”, “F” と省略できる。）“sep” オプションは区切り文字 (separator) を指定するオプションで、`sep="¥t"` はタブ区切りであることを示している。（ここで `sep=","` とすれば、区切り文字にカンマ “,” を指定して CSV ファイルを読み込むことができる。）読み込まれたデータは、データフレーム形式で保存される。最後に、`dim()` コマンドで Data オブジェクトの行数と列数（次元, dimension）を、`head()` コマンドで Data オブジェクトの最初の数行を確認しておこう。

```
> dim(Data)
[1] 100    2
> head(Data, n=2)
      x1      x2
1 -2.671  0.862
2 -2.275  1.429
```

本書で利用したサンプルデータ `appA_Rintro_Data.xlsx`, `appA_Rintro_Data.txt` では、半角英数字が使用されており以上のプログラムで問題なく R にデータを読み込むはずである。しかし、データファイルで日本語（全角文字）が使用され文字コードが Shift-JIS(ANSI) が標準である場合、文字化けが起こることがある。その

ような場合は、`fileEncoding="shift-jis"` オプションを指定する必要がある。

```
> Data <- read.table("appA_Rintro_Data.txt", header=T, sep="\t",  
+ fileEncoding="shift-jis")
```

ここまでは Windows 上での R の場合であるが、Mac OS の場合、以下のように `fileEncoding="CP932"` オプションを指定する必要がある。

```
> Data <- read.table("appA_Rintro_Data.txt", header=T, sep="\t",  
+ fileEncoding="CP932")
```

### 5.3 R から外部へのデータの書き出し

本節の最後に、今度は R 内部のデータを外部の作業ディレクトリに書き出す方法について説明する。まずサンプルデータとして、以下の例を考える。

```
> n <- nrow(Data)  
> n  
[1] 100  
> y <- runif(n)  
>  
> Data2 <- cbind.data.frame(Data, y)  
> head(Data2, n=2)  
      x1      x2      y  
1 -2.671  0.862 0.75868476  
2 -2.275  1.429 0.98027072
```

最初の `nrow()` コマンドは行 (row) 数を数えるコマンドである。(同様に列 (column) 数を数える `ncol()` コマンドもある。) `Data` は 100 行 2 列のデータフレームであるから、`n` の値は 100 である。次の `runif()` コマンドは (0, 1) の間の乱数を生成するコマンドで、引数の数 ( $n = 100$ ) だけ乱数を返す。オブジェクト `y` は、100 の乱数からなる数値型ベクトルである。次の `cbind.data.frame()` コマンドは `cbind()` コマンドに似ているが、データフレーム、行列、ベクトルなどを列方向に結合して、データフレームを作るコマンドである。新しくできたデータフレームを、`Data2` オブジェクトに保存している。`head()` コマンドで `Data2` オブジェクトの先頭 2 行を確認すると、`x1`, `x2` の隣に新しい

列  $y$  ができているのがわかる。

R 内部で生成した行列、データフレームなどのオブジェクトを外部の作業ディレクトリに書き出すコマンドは、`write.table()` コマンドである。`write.table()` の第一引数はオブジェクト名、第二引数はデータが書き込まれるファイル名で、さらにいくつかのオプションが付く。

```
> write.table(Data2, "appA_Rintro_Data2.txt", quote=FALSE,  
+ row.names=FALSE, col.names=TRUE, sep="\t", append=FALSE)
```

`write.table()` のデフォルトでは、書き出されたデータは引用符"で囲まれるが、"が必要ないときは `quote=FALSE` とする。データフレームの行名、列名を出力するか否かを制御するのが `row.names`, `col.names` オプションである。上の例では、行名は必要ないので `row.names=FALSE`,  $x1$ ,  $x2$ ,  $y$  の列名は必要なので、`col.names=TRUE` としている。`sep` オプションは出力ファイルの区切り文字を指定するオプションで、タブ区切りで出力するため `sep="\t"` としている。最後の `append` オプションは、既存のファイルに書き加えるか上書きするかを指定するオプションで、ファイルを新規作成あるいは上書きする際は `append=FALSE` とする。

## 6 その他

本節では、これまで述べてきたもの以外の個別のトピックについて述べる。

### 6.1 大文字, 小文字, 全角文字

R では、アルファベットの大文字と小文字は別のものとして扱われる。`"A"` と `"a"` は、全く別のオブジェクトである。R では、ひらがなや漢字で使われる全角文字も使用可能ではある。ただし、現在の R では全角文字を使用する際、表示に問題が起こることがあるようである。格別の理由がない限り、**全角文字は使わない**ことをお勧めする。

### 6.2 コメント

R では、「#」以降同じ行に入力された文字は、コメントとして無視される。以下の例では `#` より後にある「足し算」という言葉は、R から無視される。

```
> 1 + 2      # 足し算
```



[1] 3

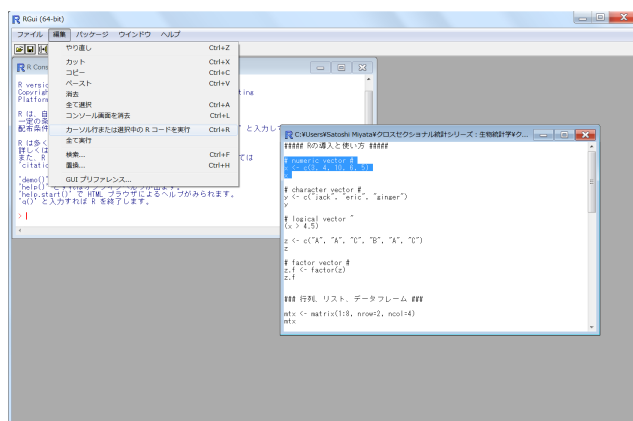
R の入力，あるいは次節に述べる R プログラムには，適宜コメントを挿入することで後で見返したとき理解の助けになる。

## 6.3 R プログラムと R Editor

これまでの説明では、R のコマンドや命令を R Console ウィンドウに直接入力してきた。しかしこの方法では、R を終了すると解析の内容が保存されないまま消去されてしまう。解析の内容を保存するためには、R に入力したコマンドや命令をプログラムの形で保存することが望ましい。そのためには、「メモ帳」などの通常のテキストエディタでプログラムを書いてもよいが、R の中で “R Editor” というものを使う方法もある。

R Editor を用いるには、「ファイル」メニューから「ファイル」→「新しいスクリプト」を選択する。すると「R Editor」ウィンドウが開く。R Editor の中には R の命令を記入することができる。(図 10)

図 10 R Editor



記入した命令を実行するには、以下の手順に従う。

1. R Editor をアクティブにする (R Editor ウィンドウの上で、マウスをクリックする)
2. 実行したい部分を、マウスで範囲選択する。
3. 以下のいずれかを実行する (どちらでも、同じ結果になる)
  - 「編集」メニューから「編集」→ 「カーソル行または選択中の R コードを実行」を選択する。

- 範囲選択した実行部分の上で、マウスを右クリックする。現れたメニューから「カーソル行または選択中の R コードを実行」を選択する。

作成した R プログラムを保存する際は、「ファイル」メニューから「ファイル」→「名前を付けて保存」あるいは「上書き保存」を選択する。R プログラムの拡張子は、必ず “\*.r” とする。このファイルは通常のテキストファイルであるので、「メモ帳」などのテキストエディタやワードプロセッサで開くことができる。R Editor において、R のプログラムを新規作成するのではなく、すでにある R プログラムを開くときは、「ファイル」メニューから「ファイル」→「スクリプトを開く...」を選び、既存の R プログラムを選択する。

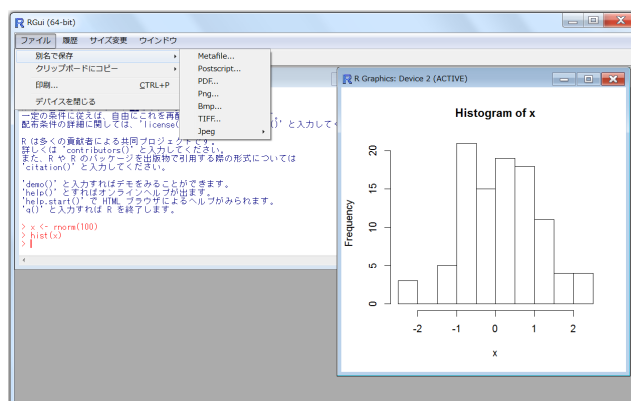
## 6.4 グラフのコピーと保存

R の中でグラフを作成した場合、R Graphics ウィンドウにグラフが出力される。レポート作成などのため、このグラフを保存したり、Word などのワープロソフトや PowerPoint などのプレゼンテーションソフトにコピーしたりしたい場合がある。グラフを保存あるいはコピーするには、以下の手順に従う。(図 11)

1. R Graphics ウィンドウをアクティブにする。(R Graphics ウィンドウの上で、マウスをクリックする)
2. 以下のいずれかを実行する(どちらでも、同じ結果になる)
  - R Graphics ウィンドウの上で、マウスを右クリックする。  
**コピー** 「メタファイルにコピー...」あるいは「ビットマップにコピー...」を選択する。  
**保存** 「メタファイルに保存...」あるいは「ポストスクリプトに保存...」を選択する。
  - 「ファイル」メニューから  
**保存** 「別名で保存」→ 保存する画像ファイル形式を選択する。  
**コピー** 「クリップボードにコピー」→ 「ビットマップとして」あるいは「メタファイルとして」を選択する。

コピーしたグラフは、Word, PowerPoint 等に張り付けることができる。保存したグラフも、同様に挿入することができる。コピーあるいは保存する画像ファイルの形式は、適宜選択する。

図 11 グラフのコピーと保存



## 6.5 拡張パッケージのインストールとロード

R の大きな利点の一つに、基本となるシステムの他に多くの研究者によって開発された統計モデルが「拡張パッケージ」として提供されている点が挙げられる。拡張パッケージは元々のシステムにはインストールされていないので、インターネットを通じてダウンロード、インストールする必要がある。

拡張パッケージをインストールするには、インターネットに接続された状態で、「パッケージ」メニューから「パッケージ」→「パッケージのインストール...」を選択する。「CRAN mirror」というウィンドウが開くので（図 12）、一番上の“0-Cloud”か日本国内のミラーサイト、その他を選択する。次に、利用可能な拡張パッケージのリスト（図 13）が表示されるので、インストールしたいパッケージを選択し「OK」を押す。

例えば“MASS”というパッケージをインストールした場合、「パッケージ ‘MASS’ は無事に展開され、MD5 サムもチェックされました」と表示されればインストールは成功である。

拡張パッケージを使用するときは、以下の二通りの方法がある。

- 「パッケージ」メニューから「パッケージ」→「パッケージの読み込み...」を選択し、「Select one」ウィンドウから使いたいパッケージを選択する。
- R Console ウィンドウで `library()` コマンドを使って、使いたいパッケージ名（例えば“MASS”）を指定する。

```
> library(MASS)
```

図 12

## パッケージのインストール

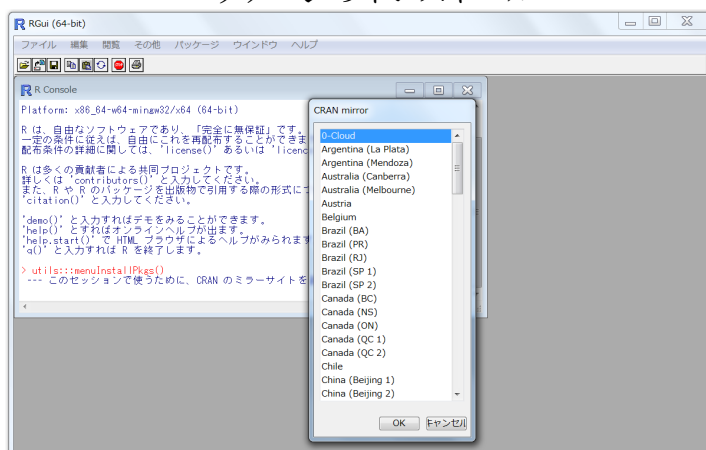
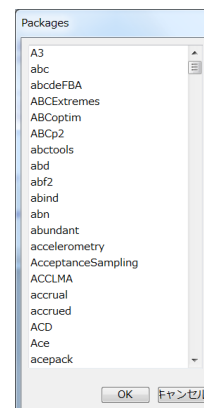


図 13



## 6.6 プロキシの設定

基本的には以上の手順により、インストールとパッケージの利用が可能なのはである。ただし、大学などプロキシが設定された環境では、`Sys.setenv()` コマンドでプロキシサーバーのサーバーアドレスとポート番号を指定する必要がある。R に入力するコマンドは

```
> Sys.setenv(http_proxy="http://*****:8080")
```

ただし、`http://*****` はサーバーアドレス、8080 はポート番号であるので、実際のアドレスはシステム管理者に確認していただきたい。

R の R Console ウィンドウの中で、コマンド名の前に “?” をつけて入力すると詳細なオンラインヘルプが得られる。英語であるが、がんばって読んでみよう。例えば、`write.table()` コマンドのヘルプを得るには、以下のように入力する。

```
> ?write.table
```

表 1 R の導入と使い方のコマンドリスト

コマンド名	目的	使い方
q()	R を終了する	「作業スペースを保存しますか？」には「いいえ」
c()	オブジェクトを結合する	c(3, 4, 10, 6, 5)
log(), exp() sin(), cos(), tan() asin(), acos(), atan()	対数関数, 指数関数 三角関数 逆三角関数	log(10), etc.
factor()	因子型ベクトルを定義する	factor(1:3) ⇒ 数値型ベクトル (1, 2, 3) を因子型に変換する
matrix()	行列を定義する	matrix(1:8, nrow=2, ncol=4) ⇒ 2 行 4 列の行列を定義する
cbind()	列ベクトルを束ねる	cbind(1:3, 5:7)
rbind()	行ベクトルを束ねる	rbind(1:3, 5:7)
list()	リスト型オブジェクトを定義する	list(1:3, c("A", "B"))
data.frame()	データフレームを定義する	data.frame(x=1:3, y=c("A", "B", "C"))
getwd()	現在の作業ディレクトリを示す	getwd()
setwd()	作業ディレクトリを変更する	setwd("C:/Rdataset")
read.table()	外部ファイルを読み込む	read.table("appA_Rintro_Data.txt", header=T, sep="¥t")
write.table()	外部ファイルに書き出す	write.table(Data2, "appA_Rintro_Data2.txt", ) quote=F, row.names=F, col.names=T, sep="¥t"))

## 参考文献

- [1] 舟尾暢男. The R Tips 第 3 版: データ解析環境 R の基本技・グラフィックス活用集. オーム社, 2016.