目次

1 インストールと初期設定

Windows XP，Vista を想定しています．他の OS でも大差は無いと思いますが細かいことは分かりません．

http://cran.md.tsukuba.ac.jp/bin/windows/base/ より最新版のインストーラをダウンロードして実行すればインストールは完了します．ただし以下の点に注意して下さい．

- Windows Vista の場合には管理者権限でインストーラを実行して下さい (右クリックメニューから「管理者として実行」).
- 「バージョン番号をレジストリに保存する」のチェックを外さないようにして下さい (R User Configration により設定を行なう場合).

次に設定です．今回は簡単のために R User Configration というツールを使用します．

http://androids.happy.nu/doc/r-tips から R User Configration をダウンロードしてきて実行します．Vista の場合は左上の「管理者に昇格...」をクリックして管理者権限になってから右の「R を初めて使うので，...」ボタンを押して下さい．

次に環境変数の設定を行います．設定する環境変数は R_USER と R_LIBS の 2 つです．R_USER には設定ファイルなどを置くディレクトリを指定しますが，R User Configration を使った場合は既に設定されていると思います．変更した場合は該当するフォルダを作成し，Rconsole と Rdevga，.Rprofile という設定ファイルを移動しておいて下さい．R_LIBS にはパッケージ等が保管されるディレクトリを指定します．指定せずとも R_USER で指定したディレクトリの下に適当に保管されますが，具体的に指定しておくと R をアップデートした際に便利です．私の設定は

- R_USER: C:/R
- R_LIBS: C:/R/win-library

です．各自で使い易いと思う場所へ設定して下さい．R User Configration から設定することも可能です．

2 基本操作

2.1 R の起動と基本操作

インストールして作成された Rgui.exe などを実行すると新しくウィンドウが開き (コマンドプロンプトなどから実行した場合は別)，次のような文章が表示されます．

```
R version 2.9.1 (2009-06-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R はフリーソフトウェアであり、「完全に無保証」です。
一定の条件に従えば、自由にこれを再配布することができます。
配布条件の詳細に関しては、'license()' あるいは'licence()' と入力してください。

R は多くの貢献者による共同プロジェクトです。
詳しくは'contributors()' と入力してください。
また、R や R のパッケージを出版物で引用する際の形式については
'citation()' と入力してください。

'demo()' と入力すればデモをみることができます。
'help()' とすればオンラインヘルプが出ます。
'help.start()' で HTML ブラウザによるヘルプがみられます。
'q()' と入力すれば R を終了します。

>
```

最後の行に表示されている”>” はここに命令をしろという記号で，プロンプトと呼びます．”>” 以降にコマンドを入力し Enter キーを叩くことで R は命令を実行します．

2.2 四則演算

R の命令はそれほど複雑なものではありません．まずは四則演算からやってみましょう．

```
> 1 + 3
[1] 4
> 8 - 6
[1] 2
> 2 * 5
[1] 10
> 4 / 2
[1] 2
```

ここで”>” はプロンプトを表しているので入力する必要はありません．[1] というのは出力結果です．数字は出力の 1 番目であることを示しています．もし出力が複数あり，1 行で表示しきれない場合は行頭に表示されているデータが何番目のものであるかが示されます．

```
> 1:20
 [1]  1  2  3  4  5  6  7  8  9 10 11
[12] 12 13 14 15 16 17 18 19 20
```

今実行したのは連続した数値ベクトルを作成する命令です．また後で解説します．

このように，R では入力された命令が即座に解釈され，結果が返ってきます．画面に出力可能なものであれば上のように出力されます (出力先は変更することもできます)．

また，基本の四則演算以外にも，べき乗，商，余りなども演算子により計算できます．

```
> 2^10                                  # べき乗
[1] 1024
> 17 %/% 3                              # 商
[1] 5
> 17 %% 3                               # 余り
[1] 2
```

ここで”#” はコメントを開始する合図です．コメントは命令の実行時には無視されます．コメントは#から改行まで続きます．R に直接命令を渡す場合はあまり有難味がありませんが，普通 R の命令はテキストファイルにまとめて書いておいて実行します．そのような場合，あとで見返しても容易に理解できるようにするため書き込むのがコメントです．未来の自分は他人と思い，なるべくまめに丁寧に書くようにしましょう．

2.3 関数

基本の算術演算よりも複雑な計算をしたり，あるいはグラフを描いたりするのには関数を利用します．R における関数の利用法は Excel などの表計算ソフトにおける関数の利用方法とほとんど同じで，” 関数名 (引数 1，引数 2，引数 3，...)” のように使用します．試しに 1〜5 までの整数の平均値，分散 (不偏分散)，標準偏差

(標本標準偏差) を計算してみます.

```
> mean(c(1,2,3,4,5))                    # 平均値は mean()
[1] 3
> var(c(1,2,3,4,5))                     # 不偏分散は var()
[1] 2.5
> sd(c(1,2,3,4,5))                      # 標準偏差は sd()
[1] 1.581139
```

すぐあとで説明しますが，c() というのも関数で，複数の数値を 1 つにまとめる働きがあります．つまり，関数は入れ子にすることも可能です.

R には膨大な量の関数が用意されています．これらを全て覚えて利用することはまず不可能ですから，頻繁に help を参照することになります．help を参照するためには?もしくは help() を利用します．例えば上記 mean() 関数の利用法を調べる場合は以下のように入力します.

```
?mean
help()
```

設定にもよりますが，普通に Windows へインストールした場合は別 Window が開いてヘルプが表示されると思います．また，用意されている場合に限りますが example(mean) などとするとその関数の利用例が表示されます.

2.4 データセットの作り方

2.4.1 ベクトルと代入

R ではベクトルという単位を基本としてデータを扱います．ベクトルを作成する方法は色々ありますが，上述した c() という関数を使用するのが最も簡単です.

```
> c(1, 2, 3, 4, 5)
[1] 1 2 3 4 5
```

c() の括弧の中にまとめたいデータをコンマで区切って列挙します．これによりベクトルが作成され出力されます.

今は数値型のデータを要素にしましたが，他に文字 (列) 型，複素数型，論理型，NULL などがあります．異なる型の要素をまとめてベクトルにした場合はより表現力の高い型にまとめて変換されます.

```
> c("a", "b", "c")                    # 文字列は""で括る
[1] "a" "b" "c"
> c("カツ丼", "天丼", 24)              # 24 は数値型だが変換される
[1] "カツ丼" "天丼"   "24"
```

また，:を使うと連続した数値ベクトルを簡単に作成できます．

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 5:3
[1] 5 4 3
> -3:1
[1] -3 -2 -1  0  1
```

この他にもベクトルを作成する関数が色々と用意されていますが，まずは上記の方法を覚えておきましょう．特に連続した数値ベクトルは繰り返しのカウントなどによく使います．

ベクトルを含め R により作成されたデータなどは全てオブジェクトと呼ばれます．作成したオブジェクトを後で利用できるようにするためには，変数への代入という操作を行います．代入操作により変数名はオブジェクトの変わりとして利用することができるようになります．

```
> x <- 1:5              # オブジェクトの代入
> x                     # 代入によりオブジェクトを後から変数名で呼ぶことができるようになる
[1] 1 2 3 4 5
```

また，ベクトルの要素へは変数名の後ろに添字を付けることでアクセスができます．添字は直接要素の番号を指定するか，論理値ベクトルを与えることができます．

2.4.2 データフレーム

R が扱えるベクトル以外のデータ形式としては，データを 2 次元に並べた行列 (see `?matrix`) や，3 次元以上に並べた配列 (see `?array`)，異なるオブジェクトをまとめたリスト (see `?list`) などがありますが，通常解析をする場合はデータフレーム (dataframe) と呼ばれる形式を多く使います．Excel などの表計算ソフトからデータを持ってきた場合も大抵は自動的にデータフレームになります．

データフレームは複数のベクトルをまとめたもので，外見は Excel のデータシートのような 2 次元配列です．1 つの列が 1 つのベクトルに対応し，各ベクトルの要素型は異なってもかまいません．データフレームは `data.frame()` 関数で作成します．

```
> a <- 1:5
> b <- c(T, F, T, T, T)                    # TRUE,FALSE は省略して書ける
> c <- c("Abc", "def", "hoge", "huga", "白米")
> ## 代入操作と同時に出力もしたい場合は式を () で括る
> (my.dataframe <- data.frame(num = a, lo = b, name = c))
  num    lo name
1   1  TRUE  Abc
2   2 FALSE  def
3   3  TRUE hoge
4   4  TRUE huga
5   5  TRUE 白米
```

上記のように，data.frame(**名前=変数名**,...) という形で列に名前を付けることができます．もし省略して data.frame(a,b,c) などとした場合には，変数名がそのまま列名として使用されます．

データフレームに含まれる各ベクトルには列名を用いて**データフレーム名$列名** とするか，**データフレーム名**［ , **列番号**］と添字を用いることでアクセスすることができます．添字を使う場合は**データフレーム名**［**行番号**，**列番号**］のように行を指定することもできます．この行番号と列番号をカンマで区切って指定する形式の添字は 2 次元配列に対するもので，番号を指定しないことはすべての行，または列を指定することと同義です．

```
> my.dataframe$num
[1] 1 2 3 4 5
> my.dataframe[ ,1]
[1] 1 2 3 4 5
> my.dataframe[2, ]
  num    lo name
2   2 FALSE  def
```

2.4.3 表計算ソフトからのデータ取り込み

Excel や Calc で入力したデータを R に取り込む方法は色々ありますが，CSV(Comma-Separated Values) 形式のテキストファイルを作成してそれを取り込むか，クリップボードにコピーしたものを取り込むのが一般的です．このとき，表計算ソフトでは上述したデータフレームの形式に沿った形で入力しておく必要があります．

上に実際の入力例[*1]を示しました．表計算ソフトで計算しやすいようにするためには，例えば品種ごとに列を作ってデータを入力し，タイトルを階層構造にしたりすると思いますが，そのようなことをしてはいけません．長さならば長さで 1 列にまとめ，品種の情報はまた別の列を作成してそこに入力します．また，正しく読み込まれない可能性があるため，2 バイト文字や括弧，特殊文字などは使わず，ローマ字と数字，コンマを混ぜて記述しましょう．

まず，CSV ファイルを読み込む方法です．先ほど示した内容が書き込まれた CSV 形式のファイルが R のワーキングディレクトリ[*2]にあるとして，次のように `read.csv()` 関数を用いてデータを読み込みます．読み込まれたデータは自動的にデータフレームになります．

```
> my.iris <- read.csv("iris.csv")
```

クリップボードから読み込む場合は，読み込みたい範囲をクリップボードにコピーした後，`read.delim()` 関数を使って読み込みます．クリップボードを指定するには`"clipboard"`と入力します．こちらもデータフレームが作成されます．

```
> my.iris2 <- read.delim("clipboard")
```

実際は `read.csv()`，`read.delim()` ともに `read.table()` という関数に適当な引数を与えた状態で実行されるラッパ関数です．詳しく知りたい場合は `read.table()` の help も参照すると良いでしょう．

`read.table()` 関数とそのラッパ関数により読み込まれたデータフレームに文字列からなる列が含まれていた場合，その列は因子型と呼ばれる型に変換されます．実際にその列が「因子」つまり品種だとか試験区だとかを表していればこれといって支障はなく，解析に都合がいいのですが，これが因子ではない場合は多少困ったことになる場合があります．因子型として読み込まれるのを抑制するには，引数 `stringsAsFactors` に `FALSE` を与えます．

```
> my.iris3 <- read.csv("iris.csv", stringsAsFactors=F)
```

[*1] 実際は R に含まれる iris というデータセットを出力したものです
[*2] getwd() で調べることができます．また setwd() で変更できます

また，引数 as.is を利用して列ごとに因子にするかしないかを指定する方法もあります．詳しくは help を参照して下さい．

2.5 統計量の計算

2.5.1 データテーブルの概要を確認する

実際のデータを元に統計量を計算してみます．サンプルデータとしては R に組み込みで用意されている iris というものを使います[*3]．iris と打ち込むとデータフレームの中身を確認することができますが，量が少々多いので head() という関数を使います．この関数はデータの先頭の方を数行だけ表示してくれる関数です．同様にデータの最後尾から数行を表示する tail() という関数もあります．

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

また，summary() という関数を利用すると各列の主要な統計量を計算してまとめたものを表示してくれます．

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

ここで数値データである Sepal.Length などは平均値や四分位点が計算されていますが，因子型ベクトルである Species では含まれる因子の種類と数が計算されています．R の関数はこのように引数として与えられる

[*3] R にはこのような組み込みのデータセットが数多く用意されています．一覧は data() で取得することができます．

オブジェクトの型により適切な処理を自動的に選んでくれるものが数多くあります.

これでデータの概要は分かったと思いますが，もっと詳細な情報が知りたい場合は?iris を参照してみると良いでしょう．組み込みのデータセットには大抵 help が用意されているので関数と同様に参照することができます.

2.5.2 データフレームに関数を適用する

最初の方で mean() や var() という関数の使用例を見ました.

```
> mean(c(1,2,3,4,5))                    # 平均値は mean()
[1] 3
> var(c(1,2,3,4,5))                     # 不偏分散は var()
[1] 2.5
> sd(c(1,2,3,4,5))                      # 標準偏差は sd()
[1] 1.581139
```

ここでは関数の引数としてベクトルを与えています．これらの関数にはデータフレームを引数として与えることもできます.

```
> mean(iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
    5.843333     3.057333     3.758000     1.199333           NA
Warning message:
In mean.default(X[[5L]], ...) :
  argument is not numeric or logical: returning NA
> var(iris)
             Sepal.Length Sepal.Width Petal.Length Petal.Width Species
Sepal.Length   0.68569351 -0.04243400    1.2743154   0.5162707      NA
Sepal.Width   -0.04243400  0.18997942   -0.3296564  -0.1216394      NA
Petal.Length   1.27431544 -0.32965638    3.1162779   1.2956094      NA
Petal.Width    0.51627069 -0.12163937    1.2956094   0.5810063      NA
Species                NA          NA           NA          NA      NA
Warning message:
In var(iris) : NAs introduced by coercion
> sd(iris)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
   0.8280661    0.4358663    1.7652982    0.7622377           NA
Warning message:
In var(as.vector(x), na.rm = na.rm) : NAs introduced by coercion
```

最初の例とは異なり，mean() と sd() では各ベクトルにそれぞれ関数が適用された結果が返ってきています．var() では各ベクトル間の共分散が計算されています．因子型ベクトルである Species ではこれらの統計量を計算できないのでエラーが返ってきています．

このように，引数として与えられたオブジェクトに応じて実際に使われる関数が決定されるものを総称的 (generic) 関数と呼びます．総称的関数があるおかげで，オブジェクトに応じていくつもの関数名を覚える必要がありません．

2.6 プロット

R ではプロットも関数により実行します．プロットのためにも非常に数多くの関数が用意されていますが，基本は主に散布図を描くために使う plot() 関数です．「主に」と付けたのは plot() もまた総称的関数だからです．大抵のオブジェクトは plot() によって適当な形にプロットされます．

最も基本的な形の散布図を描画するには 1 組のデータベクトルを引数に与えます．この場合，x 軸にデータの index が，y 軸にデータの値が取られてプロットされます．また，2 組のデータベクトルを引数に与えた場合，引数 x に与えたベクトルが x 軸に，引数 y に与えたベクトルが y 軸になります．

```
> a <- c(1,3,5,4,2)
> b <- c(5,5,3,2,1)
> plot(a)
> plot(x=a, y=b)
> plot(a, b)          # 引数名は順番さえ合っていれば省略できる
```

プロットの結果は設定にもよりますが，Windows 環境で初期設定のままならば R Graphics という名前の新しいウィンドウが開き，そこに描画されると思います．プロット結果の出力先をグラフィックデバイスと呼びます．グラフィックデバイスも関数により呼び出します．Windows 環境で自動的に開くグラフィックデバイスも windows() という関数により明示的に呼び出すことが可能です．ウィンドウが開いて描画されるもの以外にも，png や jpeg，pdf などのデバイスを選択することも可能です．利用できるデバイスの一覧は ?device を参照して下さい．windows デバイスでもプロットの保存などは可能ですが，複数の画像ファイルを自動的に作成したりするのには必ずしも適しません．状況に応じて使い分けましょう．

プロットのための関数には他にも boxpot() や barplot など様々なものがあります．また，プロット関数には多くの引数があり，設定を変更する par() 関数もあります．詳しくはこれらの関数の help などを参照して下さい．

3 R の実行環境の構築

インストールした直後の状態だと，R の実行環境としては Rterm.exe や Rgui.exe が存在していると思います．Rgui には R のスクリプトを編集して実行する最低限の機能が備わった Reditor が付属していますが，必ずしも使いやすいものではありません．

そこでここでは R を外部のエディタから利用する方法を解説します．これにも様々な方法がありますが，今回利用するのは Emacs というテキストエディタと ESS というプログラムです．

3.1 Emacs(Meadow) のインストールと設定

3.1.1 環境変数の設定

Windows 向けの Emacs 実装として Meadow というものがあります．ネットインストーラが用意されているのでインストールは簡単です．その前に環境変数を設定しておきましょう．環境変数 HOME を新たに作成し，`C:\home` などの分かりやすい場所を指定しておきましょう (`C:\home` は作成する)．Meadow に限らず環境変数 HOME は作業の起点となる場合が多くあります．大抵の場合はここに設定ファイルを置くことになります．

3.1.2 Meadow のインストール

http://www.meadowy.org/meadow/wiki/ダウンロード/からネットインストーラをダウンロードしてきます．今回インストールするのは Meadow3 です．setup-ja.exe をダウンロードして下さい．このインストーラは後でパッケージを追加したり更新する際にまた利用することになりますから，home ディレクトリなど分かりやすい場所に移動しておきましょう．

起動するとまずインストール先を聞いてきます．`C:\meadow` や `C:\home\meadow` などが分かりやすくて良いと思います．次にインストーラがダウンロードしたパッケージの保存先を聞いてきますから適当な場所を指定します．その後適当な選択肢を選択して進めていくとどのパッケージをインストールするか尋ねてきます．全部インストールしてもかまいませんが，容量と時間が必要になるのでとりあえずは最初指定されているものだけで大丈夫だと思います．

3.1.3 .emacs ファイルの作成

Meadow(Emacs) の設定は.emacs というファイルに emacs-lisp(elisp) という言語で記述します[*4].emacs ファイルは最初からあるわけではないので作成する必要があります．しかし，Windows ではシステムの関係で「右クリック → 新規作成」では作成できません．

そこで，今インストールした Meadow を使ってファイルを作成します．Windows と多少キーの使い方が異なるのでとりあえず以下のコマンドを順に入力して下さい．

- Meadow を起動
- `C-x C-f`(Find file, 「ファイルを開く」に相当するが，ファイルの作成もできる)
- 画面の下 (ミニバッファ) に `Find file: ~/bin` などと出る
- `Find file: ~/.emacs` と入力 (`~`はホームディレクトリを示す)
- `C-x C-w`(Write file, 「名前を付けて保存」に相当)
- `C-x C-c`(Meadow を終了)
- `C:\home\.emacs` が出来ていることを確認

ここで `C-x` などとありますが，これは `Ctrl` キーを押したまま `x` をタイプすることを表しています．つまり，`C-x C-f` ならば `Ctrl` キーを押したまま `c f` とタイプします．また同様の表記で `M-x` などというのも良く見掛けます．これは meta キーを押したままキーをタイプすることを表しています．これは Windows なら

[*4] elisp を覚えた方がカスタマイズの幅は広がりますが，ただ使うだけなら誰かの設定をコピペするだけでも大丈夫なので心配いりません

ば Alt キー，Mac なら Command キーになっていると思います．あるいは，Esc をタイプし，指を離して次のキーをタイプしても同様です．

3.2 ESS のインストール

ESS，つまり Emacs Speaks Statistics は S-Plus や SAS，そして R といった統計解析のためのプログラムを快適に使うための Emacs のアドオンパッケージです．

Meadow にこのようなパッケージを追加するのは簡単です．ダウンロードしてきたパッケージを解凍して，\Meadow\site-lisp ディレクトリの下へ移動させ，.emacs に読み込むための記述を追加するだけです．

ESS のダウンロードは http://stat.ethz.ch/ESS/ から行います．解凍してできたフォルダを site-lisp フォルダに移動させたら，.emacs を編集します．とりあえずは以下のように記述しておけばいいでしょう．

```
(require 'ess-site)
(setq ess-ask-for-ess-directory nil)
(setq ess-pre-run-hook
'((lambda ()
(setq default-process-coding-system '(sjis . sjis))
)))

(set-language-environment "Japanese")
(setq default-input-method "MW32-IME")
(mw32-ime-initialize)

(defun ess:format-window-1 ()          ;; ESS 起動時 window 分割の設定
(split-window-horizontally)
(other-window 1)
(split-window)
(other-window 1))
(add-hook 'ess-pre-run-hook 'ess:format-window-1)

(setq default-frame-alist              ;; 外観の設定
      (append (list '(foreground-color . "azure3")
'(background-color . "black")
'(border-color . "black")
'(mouse-color . "white")
'(cursor-color . "white")
)
default-frame-alist))
```

これには ESS 以外の設定も含まれます[*5]．R を起動するとウィンドウが 3 分割される設定になっていますが，ディスプレイサイズが小さい場合などは分割設定を消した方がいいかもしれません．

3.2.1 ESS の使用方法

ESS の使用方法については Rjpwiki の解説ページ (http://www.okada.jp.org/RWiki/index.php?ESS) を参照したほうが良いでしょう．

3.3 Emacs(Meadow) を使えるようになるために

Emacs のキーバインドは Windows ユーザにとってあまり馴染みのないもので，これを一つ一つ覚えていくのは大変な作業です．しかし，有り難いことに Emacs にはチュートリアルが用意されています (もちろん Meadow でも使えます)．Meadow でチュートリアルを実行するには，Meadow を起動した後，Help→Emacs Tutorial と選択するか，`F1 t` とタイプします．これによりチュートリアルが開始されます．最初は 30 分程度かかりますが，1 回やるだけでもかなりの操作を覚えられます．おそらく 3 回とやらないうちに大抵の操作は覚えられるはずです．

[*5] 実際，ESS を使うのが目的であれば (require 'ess-site) だけで大丈夫です．