1　R を使ってみる

1.1　インストール

R はフリーソフトです。インターネットからダウンロードして促されるままにインストールするだけで普通は問題なく動きます。RjpWiki「R のインストール」を参考にすれば問題なくできるはずです。R は色々なことができます。体系だって学ぶのも非常によいことです。Rtips のページを一通り実行してみるとよいかもしれません。以下には、統計遺伝学・数理生物学を学び始めるときに「これだけはできるとよい」ことだけを書いておきます。

1.2　起動と終了

ソフトウェアをインストールしたら、まずは起動して終了できることが基本です。コンソールと言われる画面に

```
runif(5)
```

と入力してみましょう。５つのばらばらな数字が表示されます。５つの乱数を発生させよというコマンドを入力したからです。終了するには

```
q()
```

と入力します。保存するかしないかを尋ねられますが、やめようとしている現在から再開したいなら、保存しますし、不要ならば保存しません。ワープロ文書などの場合と同じです。

1.3　コマンドの入力、コマンドの再利用

```
runif(5)
```

というコマンドを何度も使いたいとき、そのつど、キーボードをたたくのは退屈です。二つのやり方があります。一つ目は、コマンドをテキストファイルに保存しておいて、それをテキストファイル上からコピーして、R のコンソール上にペーストする方法です。二つ目は、キーボードの矢印キーの上矢印キーを押す方法です。直前に打ったコマンドが現れます。５回前のコマンドなら、上矢印キーを５回押せばよいです。この文書のコマンドの部分をコピーペーストして実行できることを確かめましょう。

1.4　作業ディレクトリとデータの読み書き

ワープロソフトを使っているときに、作成したファイルを開いたり保存したりするとき、ある特定のフォルダ (ディレクトリ) のファイルはすぐに開くことができるし、そのフォルダにならファイルの名前を指定するだけで保存することができます。R でも同様で、コンピュータ上のどこかしらのフォルダがそのようなフォルダになっています。「作業フォルダ (ディレクトリ)」と呼ばれます。ファイルからデータを読み込んだり、ファイルに出力したりするときに、どこのフォルダでやりたいかを決めることは重要なので、Rtips のサイト作業ディレクトリの変更などを用いて、実施できるようにしましょう。

2　R で計算

2.1　四則演算

加減乗除、べき乗、対数、階乗の計算は必須です。

```
5 + 3
5 - 3
5 * 3
5 / 3
5 ^ 3
exp(5)
log(5)
factorial(5)
```

```
> 5 + 3
[1] 8
> 5 - 3
[1] 2
> 5 * 3
[1] 15
> 5 / 3
[1] 1.666667
> 5 ^ 3
[1] 125
> exp(5)
[1] 148.4132
> log(5)
[1] 1.609438
> factorial(5)
[1] 120
```

2.2 値の代入・ベクトル操作・行列操作

5 + 3 とする代わりに、x=5,y=3,x+y とすると便利です。x,y を R のオブジェクト呼び、x に 5 を代入し、y に 3 を代入する、と言います。

```
x <- 5
y <- 3
x + y
```

x,y は一つの値だけを代入しましたが、複数の値を持たせることもできます。 1 個以上の値を持たせたものをベクトルと言います。

```
x <- 1:10 # 1,2,...,10 の値を代入しています
x.1 <- c(1,2,3,4,5,6,7,8,9,10) # このようにしても同じです
x.2 <- seq(from=1,to=10,by=1) # これも同じです
```

ベクトル同士の計算は簡単にできます。

```
> x <- 1:10
> y <- runif(10)
> x + y

 [1]  1.278901  2.251387  3.305585  4.415056  5.593725  6.131442  7.189644
 [8]  8.826709  9.232137 10.345212

> x * 2 # x のすべての要素が 2 倍されています

 [1]  2  4  6  8 10 12 14 16 18 20

> log(x) # これもできます

 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
 [8] 2.0794415 2.1972246 2.3025851
```

行列もよく使います。行列は 5 行 3 列の行列には、15 個の値があります。次のように作ります。

```
> a <- 5
> b <- 3
> v <- 1:15 # 15 個の値を持つベクトル
> m <- matrix(v,a,b)
> m

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
```

```
[4,]    4    9   14
[5,]    5   10   15

> m.1 <- matrix(v,a,b,byrow=TRUE) # 15 個の値の納め方が変わります。
> m.1

     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
[5,]   13   14   15

> m.2 <- matrix(v,b,a)
> m.2

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15
```

ベクトルの和、行列の要素のすべての和、行列の行の和・列の和を求められるようにしましょう

```
> x

 [1]  1  2  3  4  5  6  7  8  9 10

> sum(x) # ベクトル x の要素の和

[1] 55

> m

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15

> sum(m) # 行列 m の要素の総和

[1] 120

> apply(m,1,sum) # 行列 m の行の和

[1] 18 21 24 27 30
```

```
> apply(m,2,sum) # 行列 m の列の和

[1] 15 40 65
```

行の和・列の和を計算するのに用いた apply() 関数は、「行について…する」「列について…する」という処理をします。上の例では、「行・列」について sum しています (和を取っています)。たとえば和を取る代わりに平均値が知りたければ次のようにすればよいです。

```
> x

 [1]  1  2  3  4  5  6  7  8  9 10

> mean(x) # ベクトル x の要素の平均値

[1] 5.5

> m

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15

> mean(m) # 行列 m の要素の平均値

[1] 8

> apply(m,1,mean) # 行列 m の行の平均値

[1]  6  7  8  9 10

> apply(m,2,mean) # 行列 m の列の平均値

[1]  3  8 13
```

3 R で確率密度分布

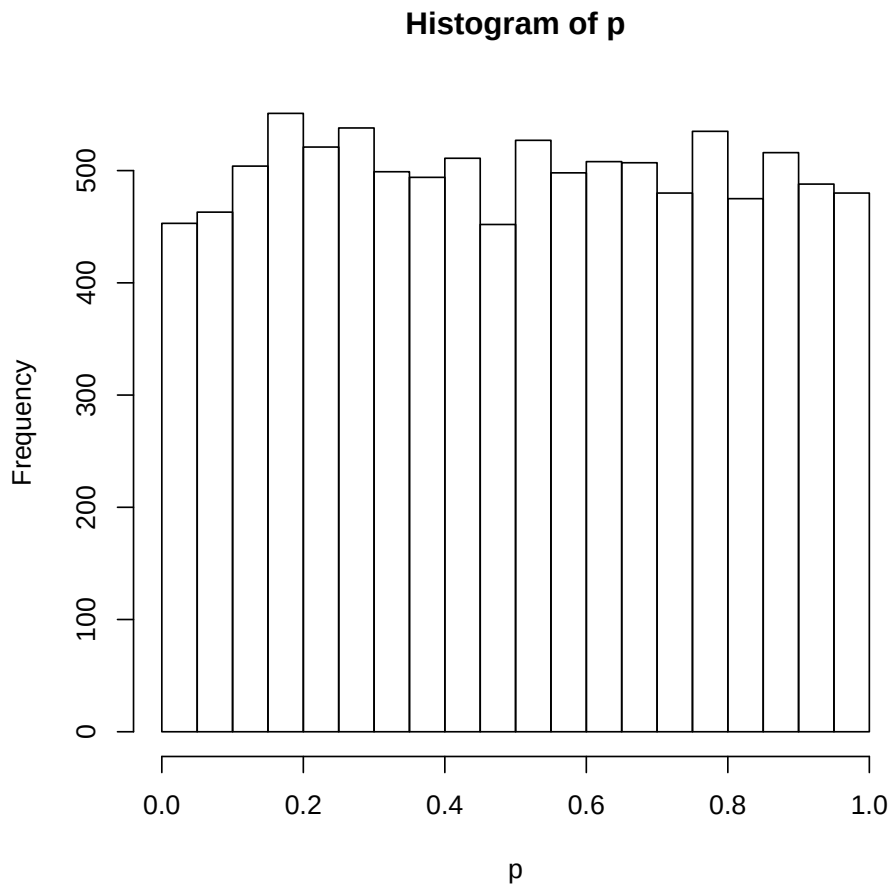
R は統計解析ソフトなので、確率密度分布の関数が使いやすいです。R の確率密分布関数の一般的な話は Rtips の確率分布と乱数を読みましょう。ここでは、いくつかの基本的なことをプロットの仕方とともに確認します。

3.1 一様分布

検定をすることを考えます。たとえば、ケース群とコントロール群とで因子の所有率に違いがあるかないかの検定のような場合です。現実には、ケース群とコントロール群とで違いがないとします。このとき、検定の

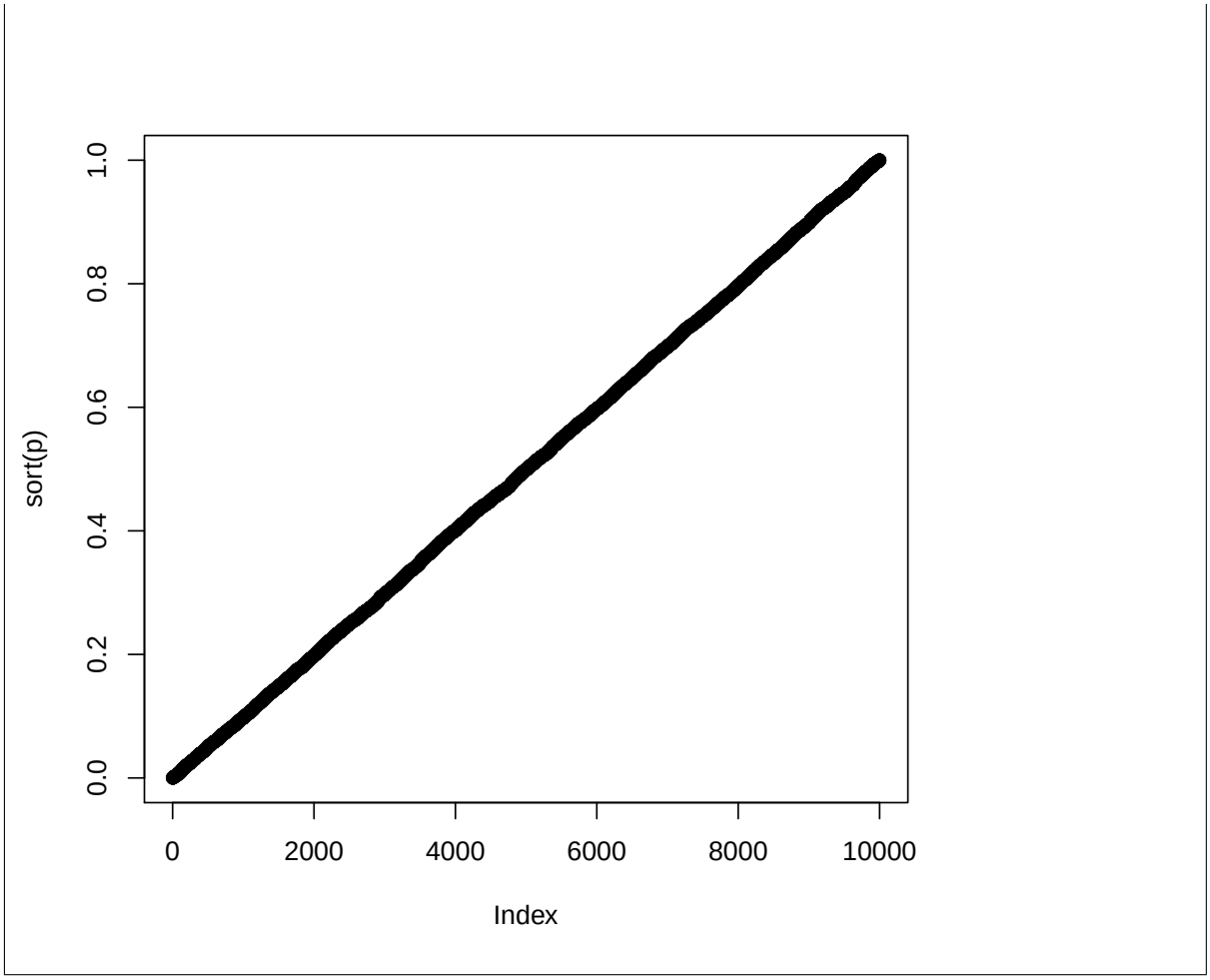
p 値は 0 から 1 までの一様分布に従います。逆に言うと、帰無仮説が成り立っているときに検定した場合に、0 から 1 までの値をまんべんなく一様に取るようなものを p 値としたとも言えます。R の runif() 関数を使って、一様分布に従う p 値を 10000 個ランダムに発生させて、その分布を描いてみることにします。

```
> n <- 10000
> p <- runif(n)
> hist(p)
```



分布が平らなので、一様に分布しているようだ、とわかります。一様に分布しているかどうかは別の方法でも視覚的に示すことができます。p 値を小さい順に並べ替えて (ソートして) プロットします。一様分布であるならば、プロットは、直線状になります。

```
> n <- 10000
> p <- runif(n)
> plot(sort(p))
```



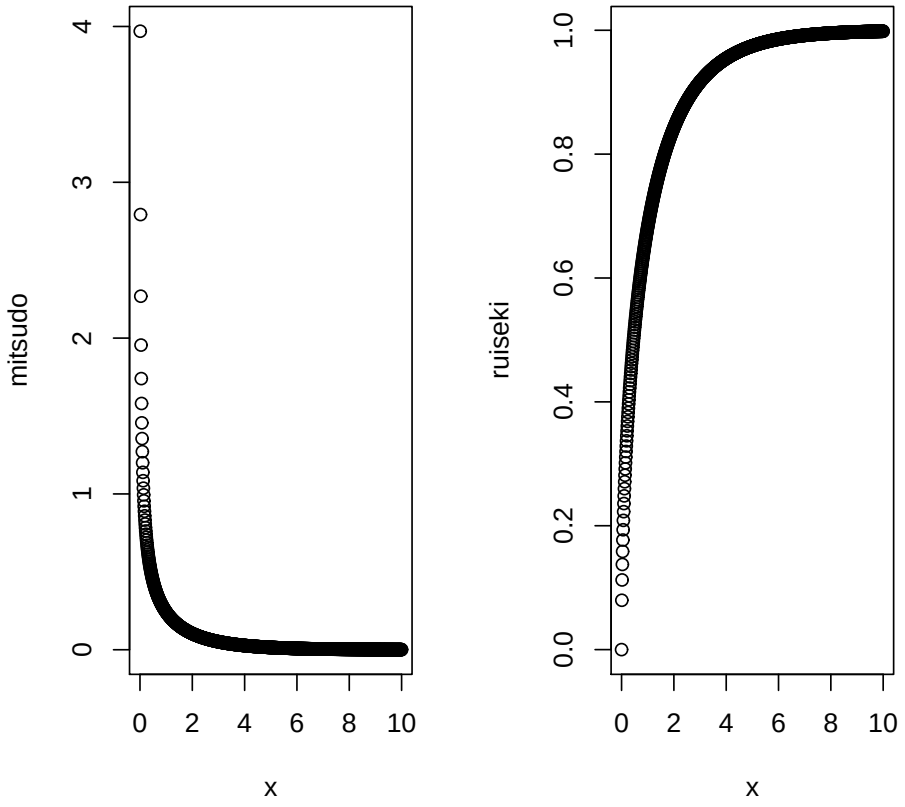
3.2 統計量と確率密度分布と累積分布

自由度 1 の χ^2(カイ二乗) 統計量を例にとります。確率密度分布を把握するには、Wikipedia の χ^2 二乗分布の記事を参照するのが有効です。記事の右側に

- 台
- 確率密度関数
- 累積分布関数

があります。台は、統計量 (χ^2) が取りうる値の範囲で、0 以上の実数であることがわかります。確率密度関数は、どんな値がどれくらいの確率で起きるかを表した関数です。台の全体について積分すると 1 になります。累積分布関数は統計量が小さい方から確率密度関数を積分して行った関数になっています。確率密度関数を台全体で積分すると 1 になるから、累積分布関数は 0 から単調に増加して 1 に収束します。これを R を使ってプロットしてみます。確率密度関数の値は dchisq() 関数で計算できて、累積分布関数の値は pchisq() 関数で計算できます。

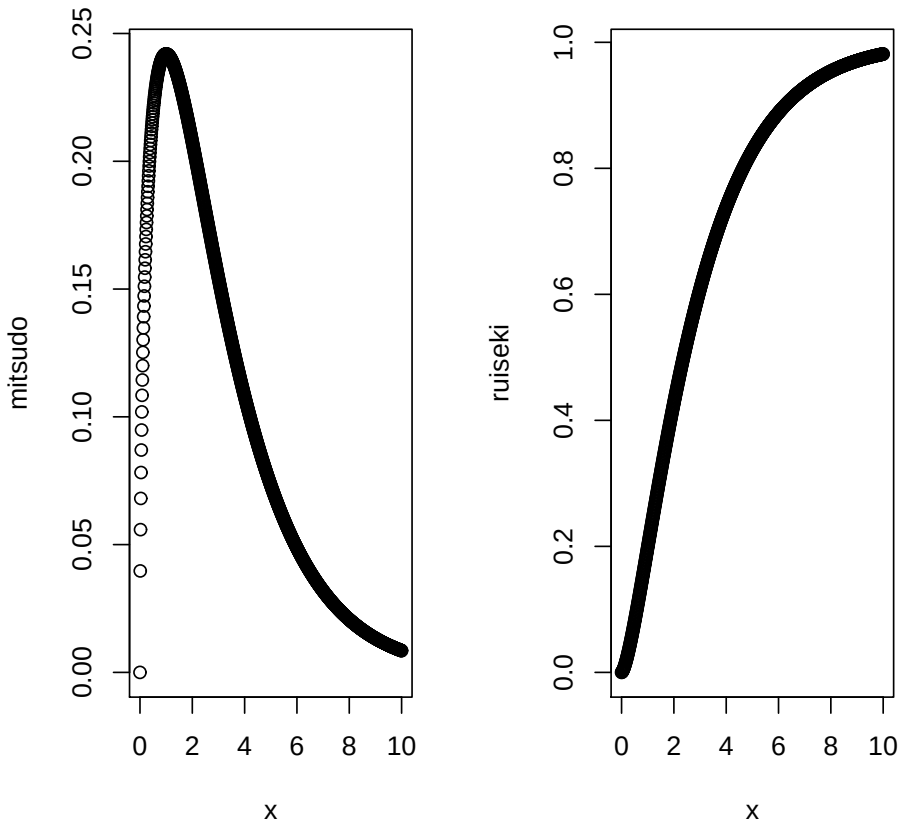
```
> df <- 1 # 自由度
> x <- seq(from=0,to=10,by=0.01) # 台を適当に決める
> mitsudo <- dchisq(x,df)
> ruiseki <- pchisq(x,df)
> par(mfcol=c(1,2)) # プロット領域を左右２分割する
> plot(x,mitsudo)
> plot(x,ruiseki)
> par(mfcol=c(1,1)) # プロット領域を１画面に戻す
```



自由度を 3 に変えて同じことをやってみます。

```
> df <- 3 # 自由度
> x <- seq(from=0,to=10,by=0.01) # 台を適当に決める
> mitsudo <- dchisq(x,df)
> ruiseki <- pchisq(x,df)
> par(mfcol=c(1,2)) # プロット領域を左右２分割する
> plot(x,mitsudo)
```

```
> plot(x,ruiseki)
> par(mfcol=c(1,1)) # プロット領域を１画面に戻す
```



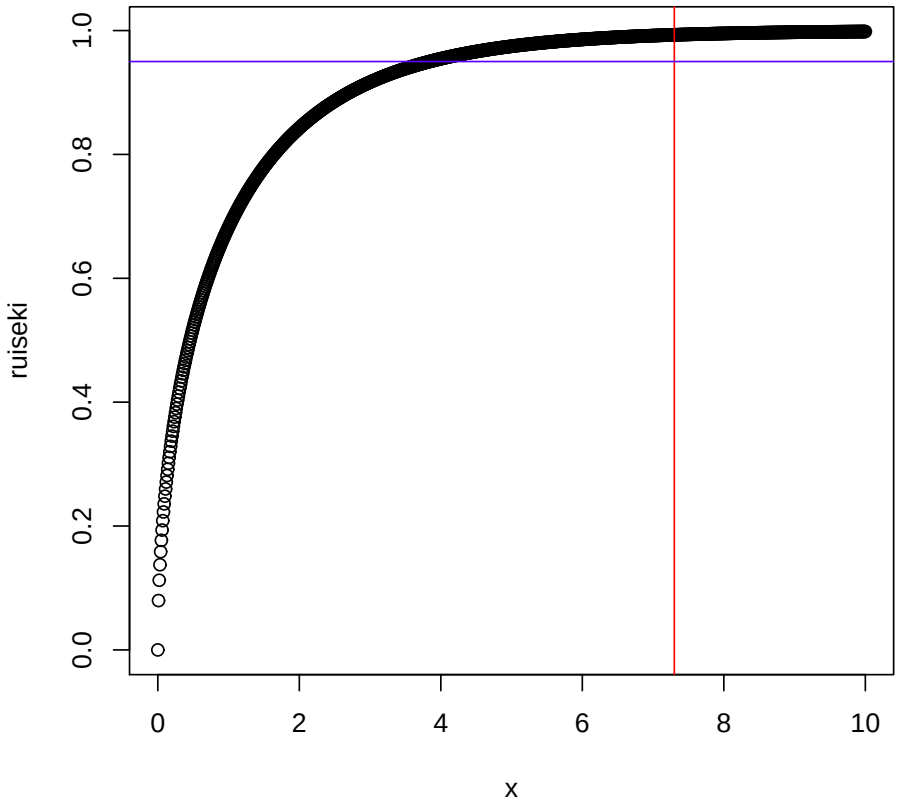
Wikipedia の絵と見比べておきましょう。

3.3 統計量と p 値

『分割表を χ^2 検定したら p 値が 0.01 より小さかったので、統計的に有意だった』というような使い方をします。これは、『分割表のセルの値から χ^2 値が計算された。分割表の自由度に照らして χ^2 値を p 値に変換したら 0.01 より小さかった』という意味です。では、自由度 1 の分割表の χ^2 が 7.3 だったときの p 値を算出してみることにしましょう。自由度 1 の χ^2 分布の累積分布で横軸の値が 7.3 のときの縦軸値が 1 よりどれくらい小さいか、が p 値になります。また、自由度 1 のときに、p 値が 0.05 になるには、いくつの χ^2 が得られる必要があるのかを計算してみましょう。χ^2 分布の累積分布で言えば、縦軸の値が 1-0.05=0.95 になっているときの横軸の値が、求めるそれです。

```
> df <- 1 # 自由度
> x <- seq(from=0,to=10,by=0.01) # 台を適当に決める
> ruiseki <- pchisq(x,df)
```

```
> plot(x,ruiseki)
> abline(v = 7.3,col=2)
> abline(h = 1-0.05,col=4)
```




```
> df <- 1
> chi2 <- 7.3
> pchisq(chi2,df,lower.tail=FALSE) # p 値が出ます

[1] 0.006895461

> p <- 0.05
> qchisq(p,df,lower.tail=FALSE) # p=0.05 のときのカイ二乗値が出ます

[1] 3.841459
```

4 関数を使う・作る

すでに sum(),mean(),runif(),dchisq() などの関数を使って来ましたが、関数とは何だったかと言うと、何か (入力値、引数) を与えると、何か (返り値) を返すものです。sum() 関数は、1 個以上の値を受け取ってその和を返し、runif() 関数は、自然数を受け取って、その数だけの乱数を返します。dchisq() 関数は、0 以上の値と自由度を指定する数を受け取り、確率密度関数の値を返します。自分でやりたいことがあり、それを何度も実行する場合には、関数を自作するのが良いです。組合せの勉強を兼ねて、次のような関数を作ってみることにします。n 個から m 個を取り出す場合の数は

$$\frac{n!}{m!(n-m)!}$$

で計算されます。$n!$ の計算には factorial(n) という関数もありますが、今は、練習の意味も含めて、

$$n! = 1 \times 2 \times ... \times n$$

であることを使います。また、1 から n までの積は n 回の積の計算の繰り返しですが、それをループという仕組みで実行することも覚えることにします。まず、関数を作る前に、$n!$ を計算してみることにします。

```
> n <- 4
> 1*2*3*4 # 検算のために答えを確認

[1] 24

> # 繰り返し掛け算をするというのは、
> # 1 に、1,2,...,n を順番に掛けあわせることだとみなします
> kotae <- 1
> for(i in 1:n){ # 1,2,...,n を順番に処理する
+         kotae <- kotae * i # 次々に掛けて大きくする
+ }
> kotae # 結果を見てみます。

[1] 24
```

うまく行っているので、これを関数にします。関数は何かを受け取って何かを返します。今回は、自然数 n を受け取って、その階乗の値を返します。受け取る値 (引数) は function(n) のように () の中に入れます。それ以外は、すべての　で囲まれた中にコピーペーストします。

```
> my.kaijyo <- function(n){
+         kotae <- 1
+         for(i in 1:n){ # 1,2,...,n を順番に処理する
+                 kotae <- kotae * i # 次々に掛けて大きくする
+         }
+         kotae # 結果を見てみます。
```

```
+ }
> # 使ってみます
> a <- 4
> my.kaijyo(a)

[1] 24
```

出来上がりました。
さて、計算したいのは

$$\frac{n!}{m!(n-m)!}$$

でした。いましがた作成した my.kaijyo() 関数を３回使って、組合せを計算してみます。

```
> n <- 10
> m <- 5
> n. <- my.kaijyo(n) # n!
> m. <- my.kaijyo(m) # m!
> n_m. <- my.kaijyo(n-m) # (n-m)!
> n./(m. * n_m.) # これが答え

[1] 252
```

計算がうまく回ったので、引数である n,m は function() の () の中に入れ、それ以外は　の中に入れることにします。

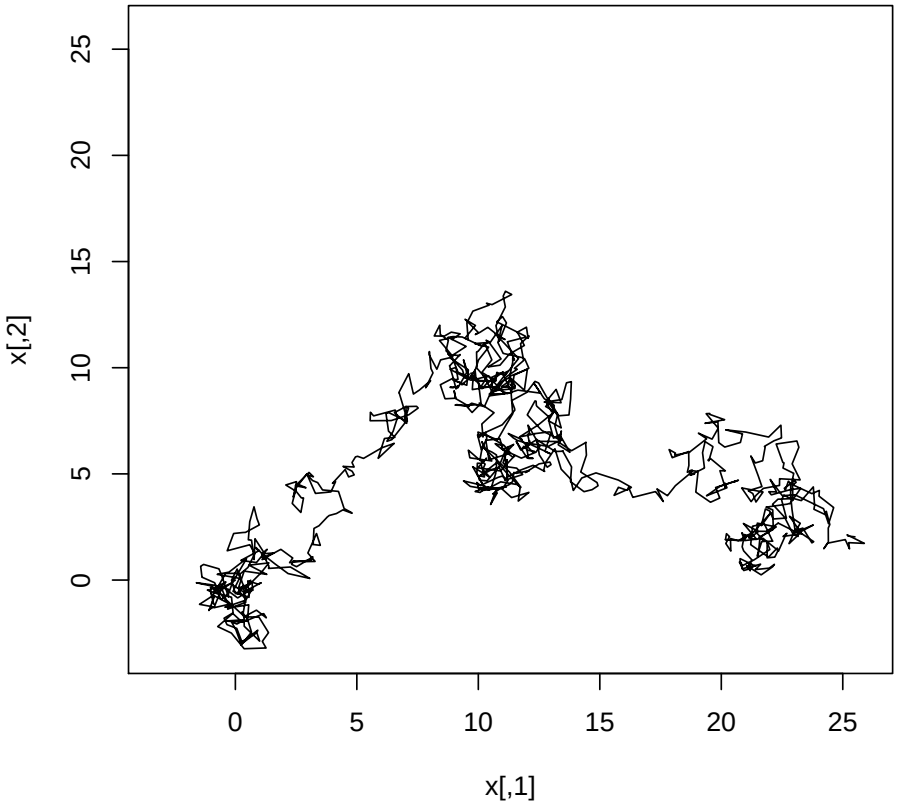
```
> my.combination <- function(n,m){
+         n. <- my.kaijyo(n) # n!
+         m. <- my.kaijyo(m) # m!
+         n_m. <- my.kaijyo(n-m) # (n-m)!
+         n./(m. * n_m.) # これが答え
+ }
> # 使ってみます
> a <- 10
> b <- 5
> my.combination(a,b)

[1] 252
```

5　R でシミュレーション

シミュレーションとしてデータを作成することが良くあります。確率密度関数からの乱数が発生できて、ループを回すことができれば、シミュレーションは可能です。

２次元の平面上をランダムに n 歩、歩いた軌跡を記録しプロットしてみましょう。

```
> # スタート地点は (0,0) の原点
> # 歩数 n
> n <- 1000
> # 全部で n+1 時点の座標を格納するのには (n+1) x 2 行列必要
> x <- matrix(0,n+1,2)
> # n 回ループ
> for(i in 1:n){
+         # １歩の歩幅を 0-1 の一様乱数とします
+         r <- runif(1)
+         # １歩の方向を 360 度=2pi、適当にします
+         q <- runif(1)*2*pi
+         # １歩のベクトルは
+         ippo <- r * c(cos(q),sin(q))
+         # １歩歩いた次の場所は、今いる場所に１歩を加えたところなので
+         x[i+1,] <- x[i,] + ippo
+ }
> # x 軸と y 軸を同じにします
> xlim <- ylim <- c(min(x),max(x))
> plot(x,type="l",xlim=xlim,ylim=ylim)
```


x[,2]
0
5
10
15
20
25
x[,1]
0
5
10
15
20
25