認知情報科学基礎実習　R Programming 1

Outline

1. 導入
2. ランダムサンプリング
3. 制御方法Flow control
4. 実習
5. 課題
6. 導入

計算モデルには、特定の数値や列・数列での効果を排除するため、乱数やランダムな数列を複数生成してモデルの性質・挙動を確認します。また、計算モデルの多くは時系列の挙動（学習曲線や分布の変化など）が扱われることが多いです。今回の実習では、ランダムな列・数列の生成方法と、時系列の挙動を生成する同一のRコマンドを反復するための制御方法を説明します。

1. ランダムサンプリング

ランダムな列の生成　― sample

*syntax*: sample(vector, n)

M個の要素を持つベクトルから、N個（N＜M）の要素をランダムに抽出する。抽出するにあたり非復元抽出とする。非復元抽出とは１度抽出したものは、次回以降は抽出の対象としない抽出方法です。

#ランダムに項目をM個選択する例：

> sample(1:10,3)

[1] 6 2 8

ベクトルは数値である必要はなく、文字でも可能です。例えば、ジャンケンの「手」をランダムに1つ生成するには以下のコマンドで可能です：

> sample(c("gu","choki","pa"),1)

[1] "pa"

列をランダムに並び替える― sample

*syntax*: sample(vector)

M個の要素を持つベクトルから、非復元抽出でM個の要素をランダムに抽出する。つまり、ベクトルの要素をランダムに並び替えることになります。

#実装例：

> sample(1:10)

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

# 抽出する数を明記しても同じです。

> sample(1:10, 10)

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

> sample(c("a","b","c","d","e","f","g"))

[1] "b" "d" "e" "c" "a" "f" "g"

非復元抽出によるランダムな列の生成― sample

*syntax*: sample(vector, n, replace=T, prob=vector)

M個の要素を持つベクトルから、復元抽出でN個の要素をランダムに抽出する。N＞Mの場合は、必ずreplace = T(復元抽出)とする必要がある。抽出にあたり、各要素が選択される確率を任意に変更することが可能となる（各要素が選択される確率の変更は非復元抽出でも適用可能です）。

# コイントスの例：

> sample(c("Head","Tail"), 10, replace=T)

[1] "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Tail" "Head" "Head" "Tail"

# バイアスのかかったコインの例（表が出る確率が0.90）：

> sample(c("Head","Tail"), 10, replace=T, prob=c(0.9,0.1))

[1] "Head" "Head" "Head" "Head" "Tail" "Head" "Head" "Head" "Head" "Head"

# サイコロを1000回ふった例：

> die <- sample(1:6, 1000, replace = T)

> head(die)

[1] 4 4 6 6 6 1

head(x)はxの要素の最初の６つを表示するコマンドです。データが正しく生成されているか、読み込まれているか確認するときに使われることが多いです。data.frameにも適用できます。

# 例

> dat<-read.csv("http://www.matsuka.info/data\_folder/datWA01.txt",header=T);

> head(dat)

shoesize h gender

1 27.0 181.4 M

2 26.5 170.8 M

3 27.5 182.3 M

4 26.5 166.8 M

5 23.5 153.2 F

6 23.0 151.6 F

1. 制御方法Flow control

Rで基礎的な制御方法（Flow control）は以下の4つになります。

FOR: 決められた回数だけ繰り返す

WHILE: 条件が満たされている限り繰り返す

IF (&ELSE): 条件が満たされたら実行する

REPEAT: 条件が満たされるまで繰り返す

2.1 **FOR** loop － 決められた回数だけ繰り返す

* *syntax*: for (*counter* in *vector*){ R commands }

counterは任意の名前を用いることができ、R commands内で参照するときに使用します。

1~5の２乗を計算し、その結果を出力する例：

for (i\_loop in 1:5){

i\_loopSQ = i\_loop^2

print(c(paste("square of", i\_loop, "is", i\_loopSQ)))

}

[1] "square of 1 is 1"

[1] "square of 2 is 4"

[1] "square of 3 is 9"

[1] "square of 4 is 16"

[1] "square of 5 is 25

上の例ではi\_loopがcounterとなり、コマンド内で使用しています（赤文字）。繰り返し数を定義しているベクトルは1:5のseqです。

繰り返し数を定義しているベクトルは数列である必要はなく、また連続している必要もありません。

str\_vec = c("a","b","c")

for (s in str\_vec){　print(s)　}

[1] "a"

[1] "b"

[1] "c"

rand\_seq = sample(1:5)

> rand\_seq

[1] 5 2 4 1 3

for (r in rand\_seq){ print(r) }

[1] 5

[1] 2

[1] 4

[1] 1

[1] 3

2.1.1 FORを用いて指数関数を算出する実習

指数関数の定義の1つとして以下の式があります：

近似値として

を、FORを用いてexp(1)算出してみましょう。

# 実装例

value = 0

for (n in 0:100){

value = value + (1^n)/factorial(n)

}

# factorial(n)はn!を算出するコマンドです

# 余談ですが、今回は実習の目的のためforを用いましたが、同じことを行うには

n = 0:100

v = sum((1^n)/factorial(n))

がより高速に計算できます。sum(x)はベクトルxの要素を全て足すコマンドです。

> x = 1:3

> sum(x)

[1] 6

2.2 **WHILE** loop － 条件が満たされている限り繰り返す

* *syntax*: while (*expression*) { R commands }

使用場面として、誤差が許容範囲になるまで計算を続ける場合や、計算し続けても数値（推定値など）が変化しない場合に終了する場合が考えられます。

# 0から５以下まで整数を繰り上げていく例

cnt <- 0

while(cnt <= 5){

print(c(paste("counter is",cnt)))

cnt<-cnt+1

}

[1] "counter is 0"

[1] "counter is 1"

[1] "counter is 2"

[1] "counter is 3"

[1] "counter is 4"

[1] "counter is 5"

#ここで用いたcntは整数を繰り上げるものです。cntを繰り上げるコマンドの位置で実行されるものが異なるので使用時には注意が必要です。上の例ではprintをしてから繰り上げ、下の例では繰り上げてからprintしています。：

cnt <- 0

while(cnt <= 5){

cnt<-cnt+1

print(c(paste("counter is",cnt)))

}

[1] "counter is 1"

[1] "counter is 2"

[1] "counter is 3"

[1] "counter is 4"

[1] "counter is 5"

[1] "counter is 6"

2.2.1 WHILEを用いて指数関数を算出する実習

Rにあるexp(1)を確認すると：

> exp(1)

[1] 2.718282

と表示されますが、実際にはより細かい数値となっています。細かい数値を確認したい場合は、format(x, digits = D)とし、xをD桁で表示することが可能です：

> print(format(exp(1), digits = 15))

[1] "2.71828182845905"

以下の式でexp(1)の近似値の誤差が0.0000000001（1e-10）未満となるまで何度繰り返しが必要か（Nの数値）検証しましょう。

#実装例：

1: err = 1

2: exp1 = 0

3: n = 0

4: while(err > 1e-10){

5: exp1 = exp1 + (1^n)/factorial(n)

6: err = abs(exp1 - exp(1))

7: n = n + 1

8: }

説明

1行目で誤差の初期化しています。Whileの中のものを実行したいので、終了条件である1e-10以上の数値であればどのような数値でも結構です。

2,3行目でexp(1)の推定値とnを初期化しています。

7行目で終了条件に必要な誤差を計算しています、abs()とは絶対値に変換するコマンドです。終了条件に関わる値や条件に関して更新は重要で、更新しないと無限ループに入ります。なおRstudioでループを強制終了するにはescキーを押すか、consoleパネルにの右上に表示されるストップサイン（赤の六角形）のボタンを押せば良いです。

なお、誤差が1e-10未満になるにはNが１４で十分でした：

> n

[1] 14

2.3 **IF (& ELSE)** control

* *syntax:* if (*expression*) { R commands }

if (*expression*) { R commands } else {R commands}

ifは上記に2つの制御とは異なり繰り返しはありません。ただし、whileと同じく、条件が満たされた場合、R commandsが（繰り返しなしで）実行されます。また、条件を満たさない場合にのみ実行したいR commandsがあればelseを用いてことによって可能となります。

age = 7

if (age == 7) {

print("you are 1-nensei or 2-nensei")

}

[1] "you are 1-nensei or 2-nensei"

年齢（age）を７歳と定義しています。次にifで７歳なら１年生か２年生と出力しています。

|  |  |
| --- | --- |
| # ifが実行される例  age = 20  if (age >= 18) {  print("you are seijin")  } else {  print("you are not seijin")  }  [1] "you are seijin" | # elseが実行される例  age = 7  if (age >= 18) {  print("you are seijin")  } else {  print("you are not seijin")  }  [1] "you are not seijin" |

2.4 **REPEAT** loop －ある条件が満たされるまで繰り返す。

*Syntax*: repeat { R commands; if (*expression*) break }# 最低でも１回は遂行される

repeat { if (*expression*) break; R commands;}# whileとほぼ同じ

WHILE と似ているが、repeat は最低でも１回は遂行されるようにできるが、while はそうとも限らない。if (*expression*) breakは終了条件を確認し（if(expression)の部分）loopを終了させる（breakの部分）コマンドです。Forやwhileにおいても使用することが可能です。

|  |  |
| --- | --- |
| # 終了条件の確認が実行後の場合（実行）  counter=6  repeat{  print(counter)  counter = counter + 1  if(counter>5){break}  }  [1] 6 | # 終了条件が実行前の場合（実行されない）  counter=6  repeat{  if(counter>5){break}  print(counter)  counter+counter+1  } |

2.2.1 REPEATを用いて指数関数を算出する実習

Whileと同様にexp(1)の近似値の1e-10未満となるまで何度繰り返しが必要か（Nの数値）検証しましょう。

err = 1

exp1 = 0

n = 0

repeat{

if (abs(exp1 - exp(1)) < 1e-10){

break

}

exp1 = exp1 + (1^n)/factorial(n)

n = n + 1

}

> n

[1] 14

1. 実習

3.1 大数の法則（乱暴に説明するとデータの数が増えると、統計量は母数に近似するというのもの）の検証。実装例は次のページにありますが、まず各自考えてみて下さい。

シナリオ：サイコロを1000回振って、６が出た割合を算出する：

　　手続きの１例

* 1. 1000回繰り返す →　forを使用
  2. サイコロの目を無作為に発生　 →　sampleを使用
  3. ６が出てきたら＋１カウント →　ifを使用
  4. N回目での割合を算出 →　forのカウンターを参照

ヒント：以下の６が出てきた割合を記録するベクトルを初期化することを推奨します。

N = 1000

prob6 = rep(0, N)

01: N = 1000

02: prob6 = rep(0, N)

03: count6 = 0

04: for (i\_trial in 1:N){

05: die = sample(1:6,1)

06: if (die == 6){

07: count6 = count6 + 1

08: }

09: prob6[i\_trial] = count6 / i\_trial

10: }

11: plot(prob6, ylim = c(0,1), xlab ="Trial", ylab ="P(die = 6)",

12: type = 'o', col = 'red',pch =20)

13: abline(h = 1/6,lty =2,lwd =2)

14: legend("topright",c("empirical", "theoretical"),

15: col =c("red","black"),lwd =2)

01~03は初期化です。

04で繰り返しを開始

05でサイコロを振る

06~08で目が６であるか確認し６である場合はcount6を繰り上げる

09でそのトライアルまでの６が出た割合を算出

11~15は以下の結果を可視化に用いた：



今回はforやifの練習として上記のような実装例となりましたが、より効率的な方法があります。その実装例は後日紹介します。

3.2 repeat/whileを用いた2つの数値の最大公約数の算出

以下は最大公約数を算出するアルゴリズムです：

1. 2つ数値を比較し、大きい値をv1、小さい値をv2とする
2. r　=　v1/v2の余りとする
3. V1をV2で置き換える
4. V2をrで置き換える
5. rが0になるまで1~4続ける

3.2.1 whileによる実装例１ v1 = 1633, v2 = 355

01: r=-99

02: v1=1633

03: v2=355

04: while (r!=0){

05: r=v1%%v2

06: print(paste('v1 =',v1,', v2 = ',v2,',remainder = ',r))

07: v1=v2

08: v2=r

09: }

[1] "v1 = 1633 , v2 = 355 ,remainder = 213"

[1] "v1 = 355 , v2 = 213 ,remainder = 142"

[1] "v1 = 213 , v2 = 142 ,remainder = 71"

[1] "v1 = 142 , v2 = 71 ,remainder = 0"

#　説明、01で余り(r)を初期化しています。０以外の数値なら何でも結構です。04で特定の条件を満たす場合に繰り返しを開始します。その他は最大公約数を算出するアルゴリズムに沿っています。rが0となった時点で、v2が０となっており（line 08）、v1が最大公約数に対応しています。最大公約数を確認すると：

> print(paste("GCD is", v1))

[1] "GCD is 71"

3.2.2 repeatによる実装例１ v1 = 1633, v2 = 355

v1=1633

v2=355

r = -99

repeat {

if (r==0){ break }

r=v1%%v2

print(paste('v1 =',v1,'v2 = ',v2,'remainder = ',r))

v1=v2

v2=r

}

[1] "v1 = 1633 v2 = 355 remainder = 213"

[1] "v1 = 355 v2 = 213 remainder = 142"

[1] "v1 = 213 v2 = 142 remainder = 71"

[1] "v1 = 142 v2 = 71 remainder = 0"

> print(paste("GCD is", v1))

[1] "GCD is 71"

3.3 バブルソート（sort:並び替え）の実装

バブルソートの説明は以下のサイトを参照して下さい：

https://ja.wikipedia.org/wiki/バブルソート

<http://www.ics.kagoshima-u.ac.jp/~fuchida/edu/algorithm/sort-algorithm/>

下に実装例があります。例を見る前に各自挑戦して下さい。

# 実装例

01: vec = c(8,4,3,7,6,5,2,1)

02: len\_vec = length(vec)

03: for (loop1 in 1:(len\_vec-1)){

04: for (loop2 in 2:(len\_vec - loop1 + 1)){

05: if (vec[loop2] < vec[(loop2 - 1)]){

06: temp\_num = vec[loop2]

07: vec[loop2] = vec[(loop2-1)]

08: vec[(loop2-1)] = temp\_num

09: }

10: }

11: print(paste("result after loop1 = ", loop1))

12: print(vec)

13: }

[1] "result after loop1 = 1"

[1] 4 3 7 6 5 2 1 8

[1] "result after loop1 = 2"

[1] 3 4 6 5 2 1 7 8

[1] "result after loop1 = 3"

[1] 3 4 5 2 1 6 7 8

[1] "result after loop1 = 4"

[1] 3 4 2 1 5 6 7 8

[1] "result after loop1 = 5"

[1] 3 2 1 4 5 6 7 8

[1] "result after loop1 = 6"

[1] 2 1 3 4 5 6 7 8

[1] "result after loop1 = 7"

[1] 1 2 3 4 5 6 7 8

# 説明

01ではwikiにある動作例と同じものを使用しています。02のlength(x)でベクトルxの要素数（ここでは８）を算出しています。05~09で２値を比較し条件を満たせば位置を転換しています。Rでは直接転換することが出来ないので、一時的にtemp\_numに一方の数値を格納しています。

15の要素を持つベクトルへの応用

vec = sample(1:15)

len\_vec = length(vec)

for (loop1 in 1:(len\_vec-1)){

for (loop2 in 2:(len\_vec - loop1 + 1)){

if (vec[loop2] < vec[(loop2 - 1)]){

temp\_num = vec[loop2]

vec[loop2] = vec[(loop2-1)]

vec[(loop2-1)] = temp\_num

}

}

print(paste("result after loop1 = ", loop1))

print(vec)

}

ここでは変更があるのはvecの値＆要素数のみです。

以下、結果：

[1] "result after loop1 = 1"

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

[1] "result after loop1 = 2"

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

[1] "result after loop1 = 3"

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

[1] "result after loop1 = 4"

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

[1] "result after loop1 = 5"

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

[1] "result after loop1 = 6"

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

[1] "result after loop1 = 7"

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

[1] "result after loop1 = 8"

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

[1] "result after loop1 = 9"

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

[1] "result after loop1 = 10"

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

[1] "result after loop1 = 11"

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

[1] "result after loop1 = 12"

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

[1] "result after loop1 = 13"

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

[1] "result after loop1 = 14"

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

Loop1 = 9で並び替えが終わっています。ですので、それ以降計算する必要はありませんでした。

4.課題（暫定的）

実習３.３で紹介したバブルソートの実装例は非効率でした（９回目後は計算する必要がないが、計算している）。上記の実装例を基に、順番に並んだ時点（位置の転換がない場合）で計算を終了するRのコードを書いて下さい。