付録.MATLAB の簡単なご案内 p.194-p.206 担当：粕谷

2008.5.11

担当者注 MATLAB の説明ー数値計算で（分野にもよるが）非常によく使われるソフトウェア。手続き型である。参照がない、ベクトルなどのインデックスが 1 からに固定されているなどの”癖”がある。クローンとして scilab や octave がある。この付録の内容は、数値計算用ソフトウェアのごく簡単な入門なので、ベクトルや行列を演算する命令を備えている数値計算用のソフトウェアを使ったことがある人は読む必要はないと思う。たとえば、R や Splus でもいいし、Mathematica の数値計算の機能でもかなりカバーされている。以下の説明では、R での対応するコードも書いてみた。

レッスン１　基本の前

MATLAB の Mat は数学ではなく行列。“ Matrix Laboratory ”が語源。MATLAB は基本的には電卓

［内容］3+4 は 7
｛R では｝

```
> 3+4
[1] 7
```

［内容］変数に値を入れておいて操作できる
｛R では｝

```
> x<-3+4
> x
[1] 7
> y<-x+2
> y
[1] 9
```

　変数（たとえば x）の中身は、１つの値ではなく、複数の値つまりベクトルでもよく、行列でもいい。
［内容］変数にベクトルを入れることができる。ベクトルの全要素への操作（演算）は個々の要素について書かなくても、ベクトルへの操作として書くことができる

｛R では｝

```
> x<-1:5
> x
[1] 1 2 3 4 5
> x^2
[1]  1  4  9 16 25
```

セミコロンの使い方《MATLAB 独特で、;を最後につけると出力が抑えられ結果が画面に出ない》。

［内容］0 から 1 まで 0.1 刻みの値からなるベクトルを x に入れる（p.195 下）

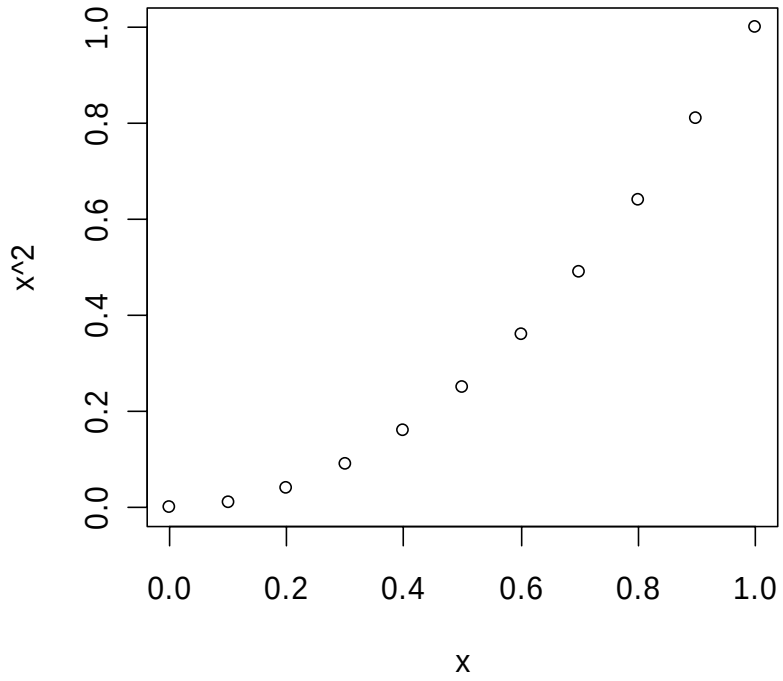
｛R では｝

```
> x<-seq(0,1,by=0.1)
> x
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

［内容］0 から 1 まで 0.1 刻みの値に対してその 2 乗をプロットする（p.195 最後）

｛R では｝

```
> y<-x^2
> plot(x,y)
> plot(x,x^2)
```



［内容］変数名は、あとで理解しやすいよう、x とかではなくもっとわかりやすいものも可。

｛R では｝

```
> activitylevel<-seq(0,1,by=0.1)
> cost<-activitylevel^2
> # this plots cost against activitylevel
> plot(activitylevel,cost)
```

レッスン２　関数

関数は引数（argument、ひきすう）に対して何かをする。すでに plot という関数が登場ずみ。いくつかの数の中から最大値を探したり（適応度の最大化で便利かも）、ある値が 0 になる場合を探したり（安定化選択を考えるとき便利かも）、チョウの分散をシミュレートしたりといったことができるだろう。MATLAB では自分で関数を作って、自作の関数と MATLAB に用意されている関数を一緒に使える。

　何かを繰り返してするときは関数が有効な手段。グラフをプロットすることも例の 1 つで、だから plot という関数が準備されている。ある環境の条件（パラメーター）の下での平衡遺伝子頻度をいろいろな場合について求めるときも、平衡遺伝子頻度を求める手順を関数にしておけば便利である。
　まずあまり役にはたたない例からはじめる。ある学生が月末にどれだけお金が残っているかを計算する。この学生はアルバイトで収入を得て、部屋代を払い、毎日ビールを飲む《以下お金の単位はユーロ、€》

［内容］関数の書式は、function[output]= functionname[arguments]
｛R では｝functionname<-function (arguments)

［内容］収入と部屋代、毎日のビール代（1 月は 30 日）から月末の残高を計算する、関数名は beerproblem
｛R では｝

```
beerproblem <-function(salary,rent,beermoney){
# beerproblem<-(salary-rent-30*beermoney)
# This calculates
# I assume
return (salary-rent-30*beermoney)
}
```

%で始まる行はコメント（注釈）である。実行されない。R では#である。MATLAB では、help　関数名　と入力すると、関数の定義の最初の%で始まる（一連の）行が出力される。定義した関数 beerproblem は、beerproblem.m という名前のファイルとしてセーブしておく。《R では、関数名.R が普通》

［内容］収入 700、部屋代 250、毎日のビール代 30 だと、月末には 450 赤字
｛R では｝

```
> beerproblem(700,250,30)
[1] -450
```

［内容］これでは赤字なのでもっと働いて収入を 1000（部屋代 250、毎日のビール代 30）にする、なお赤字
｛R では｝

```
> beerproblem(1000,250,30)
[1] -150
```

［内容］まだ赤字なので部屋代 150 の安いところに移る（収入を 1000、毎日のビール代 30）にする、なお赤字
｛R では｝
> beerproblem(1000,150,30)
[1] -50

［内容］まだ赤字なので毎日のビール代を 25 に減らす（収入を 1000、部屋代 150）にする、やっと黒字
｛R では｝
> beerproblem(1000,150,25)
[1] 100

［内容］では毎日のビール代をどれだけにすると収支は差し引き 0 になるのか（収入 1000、部屋代 150）、グラフを描いてみてみる。
linspace は等間隔に値を作る関数《R では seq でできる》
関数にすると中間の結果は残らない。中間的な結果があると紛らわしいことが多い《そうともいえないと思う》

｛R では｝
```
> beer<-seq(25,30,length=100)
> savings<-beerproblem(1000,150,beer)
> plot(beer,savings)
```
ちなみに beer の中味は
```
> beer
  [1] 25.00000 25.05051 25.10101 25.15152 25.20202 25.25253 25.30303
25.35354 25.40404
 [10] 25.45455 25.50505 25.55556 25.60606 25.65657 25.70707 25.75758
25.80808 25.85859
（中略）
 [91] 29.54545 29.59596 29.64646 29.69697 29.74747 29.79798 29.84848
29.89899 29.94949
[100] 30.00000
```

気が変わって、途中の結果たとえば、収入－部屋代も見たいとする。
［内容］収入－部屋代をまず計算し、さらに毎日のビール代も計算に入れたと

差し引きの収支を計算する。
｛R では｝

```
beerproblem2 <-function(salary,rent,beermoney){
# MBB<-(salary-rent)
# moneyleft<-(salary-rent-30*beermoney)
# This calculates
# I assume
MBB<-(salary-rent)
moneyleft<-(salary-rent-30*beermoney)
return (moneyleft,MBB)
}
```

でもできるが

```
beerproblem2 <-function(salary,rent,beermoney){
# MBB<-(salary-rent)
# moneyleft<-(salary-rent-30*beermoney)
# This calculates
# I assume
MBB<-(salary-rent)
moneyleft<-(salary-rent-30*beermoney)
return (c(moneyleft=moneyleft, MBB=MBB))
}
```

下の方を使うと

```
> beerproblem2(1000,150,25)
moneyleft       MBB
      100       850
> x1000<-beerproblem2(1000,150,25)
> x1000["moneyleft"]
moneyleft
      100
> x1000["MBB"]
MBB
850
```

　プログラムは関数にしないで、ファイルに保存しておいて使うときに呼び出して実行してもいい。この場合、スクリプトと呼ぶ《簡易的なプログラミング言語で書かれたプログラムならそう呼ばれることが多い》。ある温度のときの

ESS を求める関数 find_ess(temperature)を既に書いていたとする。例（p.200 上）のようなプログラムで、2 度から 12 度までの範囲の温度での ESS を表示できる。
｛R では｝find_ess(temperature)が仮想的な関数なので例はなし

レッスン３　行列とベクトル

MATLAB ではすべてが行列。１つの数値つまりスカラーは 1×1 の行列。
［内容］ベクトルを代入し、個々の要素を取り出す。
｛R では｝

```
> y<-c(2,4,3)
> y[2]
[1] 4
```

［内容］ベクトルを代入し、個々の要素を取り出す。
｛R では｝

```
> tail(y,n=1)
[1] 3
```

n はいくつの要素を表示するかを指定している、n=3 とすると、最後の 3 つが表示される。

［内容］行列を作ってみる
｛R では｝

```
> A<-matrix(c(2,4,3,5,7,2),ncol=3,byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    5    7    2
```

［内容］行列 A の要素(1,2)を表示
｛R では｝

```
> A[1,2]
[1] 4
```

［内容］,数字や数字,で、1 つの要素ではなく１つの行や列全体を指定することができる。
｛R では｝

```
> A[,1]
[1] 2 5
```

［内容］行列 A の２つの行（列）をつないで、行列を作る。
｛R では｝

```
> B<-cbind(A[,3],A[,1])
> B
     [,1] [,2]
[1,]    3    2
[2,]    2    5
```

［内容］行列の１つの要素だけの値を変える。
｛R では｝

```
> B[1,1]<-0
> B
     [,1] [,2]
[1,]    0    2
[2,]    2    5
```

［内容］長方形の形でない行列を作ろうとするとエラーになる。（p.201 最後）
｛R では｝

```
> B<-cbind(A[2,],A[,1])
Warning message:#以下省略
```

たとえば、ある年の、齢ごとの鳥の個体数を Birds という行列に入れるとする。Birds(:,34)は、34 という年にいたすべての齢の鳥の個体数となる。ただし、MATLAB では要素の番号（インデックス）が 1 からはじまることに注意。
　X(1,10)と書いたとき、1 や 10 は要素の番号（インデックス）である。MATLAB には要素の番号を返す関数がいろいろある。たとえばベクトルや行列の中での最大値を探す関数である max である。
［内容］形質が 0 から 2 のあいだで変化し、x が小さいと適応度は低いが大きいとコストが大きいので、適応度は x*exp(-x)となっているとする。グラフを描いて適応度が最大となる x を探す。（p.202 中）

｛R では｝

```
> x<-seq(0,2,by=0.01)
> fit<-x*exp(-x)
> plot(x,fit)
```

［内容］形質が 1 付近で最大になるようだが、もっとちゃんと探す。

｛R では｝

```
> max(fit)
[1] 0.3678794
> which.max(fit)
[1] 101
> x[101]
[1] 1
> fit[101]
[1] 0.3678794
> x[which.max(fit)]
[1] 1
```

x=1 のときに最大（この刻みでは）

レッスン４　MATLAB での真と偽

［内容］2>3 は偽

｛R では｝

```
> 2>3
[1] FALSE
```

［内容］3 つの状況での、行動 A と B の適応度はそれぞれどちらが大きいか

｛R では｝

```
> fitnessA<-c(2.3,3.4,3.3)
> fitnessB<-c(0.4,3.6,3.7)
> fitnessA>fitnessB
[1]  TRUE FALSE FALSE
```

｛R では｝

```
> DoA<-fitnessA>fitnessB
> DoA
[1]  TRUE FALSE FALSE
> BestFitness<-DoA*fitnessA+(!DoA)*fitnessB
> BestFitness
[1] 2.3 3.6 3.7
```

《R では演算順序の仕様がちがうので、カッコが必要である。ないと、

```
> BestFitness<-DoA*fitnessA+!DoA*fitnessB
> BestFitness
[1] 2.3 1.0 1.0
```

となってしまう。

```
> !DoA*fitnessB
[1] FALSE  TRUE  TRUE
```

だからである。》

レッスン５　ループ

［内容］1 から 10 までの整数の 2 倍と半分を計算

{R では}

```
> double1<-numeric(10)
> half1<-numeric(10)
> for (i in 1:10){
+     double1[i]<-i*2
+     half1[i]<-i/2
+ }
> double1
 [1]  2  4  6  8 10 12 14 16 18 20
> half1
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

#R では double は別に意味がある（倍精度）ので名前変更

#R ではループ内に始めて出てくるベクトル・行列はあらかじめ大きさを与えておかないとエラーになるー最初の 2 行

　このループは i のすべての値について何かをする。具体的に言うと、1 から 10 までのすべての i（整数）について、i の 2 倍と i の半分を計算する。もちろ

ん、この例はばかばかしいもので、この場合には（できる限りは）ベクトル計算で描くべきである。その方が《短く書けて読みやすく》速い。
［内容］すぐ上をベクトルの演算で書き直したもの
｛R では｝

```
> x<-seq(1:10)
> double1<-x*2
> half11<-x/2
> double1
 [1]  2  4  6  8 10 12 14 16 18 20
> half1
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

すでに double[]に 10 個より多くの値がはいっているとこの計算（上記のいずれも）ではそのうち最初の 10 個だけが書き換えられる。それを防ぐためには、double を初期化してクリアしておく。
｛R では｝さきほどの　> double1<-numeric(10)

　ループは遅いのでなるべくベクトル化すべきだが、ループが必要なこともある。生態学の例でよく起こるのは、その時までその値がどうなるかわからない場合である。たとえば、時刻 t+1 における個体数は時刻 t の個体数がわからないとわからない、など。

［内容］瞬間増殖率が平均 0 で標準偏差 1 の正規乱数になるような指数的増殖する個体群サイズの変化（最初の個体数は 100）
｛R では｝

```
N<-numeric(101)
N[1]<-100
r<-numeric(100)
lambda<-numeric(100)
for (t in 1:100){
      r[t]<-rnorm(1,mean=0,sd=1)
      lambda[t]<-exp(r[t])
      N[t+1]<-N[t]*lambda[t]
}
> N<-numeric(101)
> N[1]<-100
```

```
> r<-numeric(100)
> lambda<-numeric(100)
> for (t in 1:100){
+ r[t]<-rnorm(1,mean=0,sd=1)
+ lambda[t]<-exp(r[t])
+ N[t+1]<-N[t]*lambda[t]
+ }
> plot(1:101,N)
> hist(lambda,breaks=20)
```

最後の 2 行は

・t が 1 から 100 まで変化すると、N[2]から N[101]までが計算される。
・t にはベクトルは入っていないので、plot(t,N)とするわけにはいかない
・hist は MATLAB に組み込まれている関数
・ r と lambda は初期化してクリアにしておいた方がいいかもしれない《R の例ではそうしている》。

上記の例で t にあたるものが整数でない場合、たとえば、100 の異なった年ではなく、100 の異なった条件における動物の最適行動といった場合には、上記のプログラムの書き方をそのまま使えない。《２つの問題がある、for ループで整数でないループ変数が使えるか、ベクトルのインデックスに整数以外が使えるか》（p.205 最後）

［内容］0 から 1 までの 0 の実数で表される条件ごとの行動を計算し、行動を条件に対してプロットする（条件は 0.01 間隔で計算）。（p.206 上）
｛R では｝

```
> condition<-seq(0,1,by=0.01)
> behaviour<-numeric(length(condition))
> for (i in 1:length(condition)){
+ behaviour[i]<-condition[i]で計算される何か
+ }
> plot(condition,behaviour)
```

この場合もベクトル化すべき。
ループの中に別のループを置くことも可能。入れ子にした（ネストさせた）

という。
［内容］ループを入れ子にした（ネストさせた）例。0 から 1 までの 0 の実数で表される条件ごとに、２つの遺伝子型の適応度を計算する。この例では、適応度＝遺伝子型の値×条件の値とする。（p.206 中）
｛R では｝

```
> condition<-seq(0,1,by=0.01)
> genotype<-c(1,2)
> fitness<-mat.or.vec(length(condition), length(genotype))
> for (i in 1:length(condition)){
+for (j in 1:length(genotype)){
+fitness[i,j]<-condition[i] *genotype[j]
+}
+}
```

mat.or.vec()はすべての要素が 0 の行列やベクトルを作る R の関数。

この場合もベクトル化すべきである（ベクトル化については Rice,2004 の第 7 章参照）。
［内容］上記のベクトル化。（p.206 下）
｛R では｝

```
> condition<-seq(0,1,by=0.01)
> genotype<-c(1,2)
> fitness<-genotype%o%condition
```