R 講習会

飯島勇人[*†]

目次

1 **はじめに** 1
1.1 データの用意の仕方 2
1.2 データの読み込み 2

2 FIRST STAGE 5
2.1 データ操作 5
2.2 簡単な作図 11

3 SECOND STAGE 18
3.1 データの操作方法 18
3.2 作図 27

4 THIRD STAGE 40
4.1 配列の扱い 40
4.2 実習 1：複数の図の作成 46
4.3 実習 2：シカ密度分布図 51

1 はじめに

今回の講習会では、以下の点に焦点を絞ってお話をしたいと思います。

- R でデータを自在に操作できるようになる
- R で自在に作図できるようになる

* 山梨県森林総合研究所森林保護科研究員（注！（独）森林総合研究所とは一切関係ありません）
† 連絡先: 0556-22-8005 または iijima-akks@pref.yamanashi.lg.jp

1.1 データの用意の仕方

1.1.1 R に読ませるデータ

- ゑくせるなどにデータを打ち込む。
- 各列に個体番号、環境条件などを入れる。
- 1 行目にはデータのラベルを入力する。
- 日本語、特殊文字（% や○×など）は**絶対に使わない**。入力していいのは、数字、半角英文字、空欄のみ（生残なら 0 と 1 で区別する）。
- 最近の R は基本的に日本語があっても問題なく動作しますが、肝心なところでエラーの原因になったりします。できるだけ日本語（2 バイト文字）は使わない方がいいでしょう。
- 欠損の場合は NA を入れ、基本的に空のセルを作らない。
- 違った形式（並べ方が異なる）のデータを同一ファイル内に混在させない。
- ゑくせるにデータを入力したら、保存するときに「csv 形式で保存」を選び、csv ファイルとして保存する。

1.1.2 データを入力したファイルを置く場所

結論から言うと、どこでも構いません。というのは、R を立ち上げた後、ファイルがどこにあるかを指定することができるからです。指定する方法は、以下の 2 つです。

- 「ファイル」→「ディレクトリの変更」で、データのファイルがあるフォルダを選択
- `setwd("C:/`**データのファイルがあるフォルダへのパス**`/")` を入力。

1.2 データの読み込み

1.2.1 今回使うデータ

今回使うデータは、ニホンジカのモニタリグデータと（推定）個体数、捕獲数が示されているデータです。

```
 Year mesh      lda        fhc Effort  SD    Route  PG      Area  BC  DA HC x y
1 2008    1 4.673596 0.08510851      3   6 4.810494  69 1.0691115  20 107  9 1 1
2 2008    2 4.117214 0.14009740     48 127 5.086193  43 0.9799341 119  61  9 2 1
```

Year　調査年
mesh　調査メッシュ（場所）
lda　対数シカ個体数
fhc　捕獲率
Effort　出猟人日

SD　シカ目撃数
Route　踏査距離（糞塊調査）
PG　糞塊数
Area　調査面積（区画法）
BC　シカ目撃数
DA　シカ個体数
HC　シカ捕獲数
x　メッシュの x 座標
y　メッシュの y 座標

1.2.2 データの読み込み方

R にデータを読み込むには、`read.csv()` という関数を使います。

```
> d <- read.csv("data.csv")
```

このとき、特にエラーなどが出なければ、きちんと R にデータは読み込めています。

読み込めている、ということをもう少し正確に説明します。R では、元のデータファイルそのものを読み込んでいるわけではありません。データファイルの内容のみを `read.csv()` という関数で読み込んで、上記の例では d という「オブジェクト」に保存しています。

オブジェクト、とは聞きなれない言葉ですが、要はデータが格納されている物、とでも思ってください。オブジェクトは、

- 数字が先頭
- 数字のみ

でなければ、どんな名前でも構いません。

逆にエラーが出てしまった人は、以下の 2 点を確認して、もう一度やり直してみてください。

- 上記のコマンドの打ち込み間違い（これがすごく多い）
- データファイルがある場所をきちんと認識させられているか（`getwd()` と打つと、R が認識している現在のフォルダが表示されます）

1.2.3 知っておくと便利な小技

- キーボードの上矢印（↑）を押すと一つ前に打ったコマンドが、下矢印（↓）を打つと先のコマンドが表示されます。
- R に直接命令を出す（コマンドを打つ）のではなく、一度何かしらのファイルにコマンドを下書きし、それをコピペで R に貼り付けることでコマンドを実行しましょう。特に作図ではコマンドが長くなるので、こうしておかないと結構絶望的なことになります。

- `1:10` というように数字の間に:をはさむと、左の数字から右の数字まで 1 ずつ変化する数列を生成できます（等差数列)。
- 画面で表示される数字の桁数を変えたい場合は、`options(digits=`**小数点以下の桁数**`)` と入力します。
- 画面で表示される数字について、指数表現（1.0e-2）にさせやすくする場合は `options(scipen=`**負の値**`)` と入力し、逆にさせにくくする場合は `options(scipen=`**正の値**`)` と入力します。

2 FIRST STAGE

この節では、簡単なデータ操作と作図の方法を扱います。

2.1 データ操作

2.1.1 概要を表示させる関数

R の画面は大きくない上に、場合によっては同じ行のデータがずれて表示されてしまったりするので、そのままではデータを見るのに適していません。データを見たいというときには、元のデータファイルを立ち上げるという手もありますが、以下のような関数が役に立ちます。

- head(**オブジェクト**, **行数**)：データの先頭〜行分を表示させる。行数の指定は省略することもできる。
- summary(**オブジェクト**)：データの値の要約値を表示させる。

head() の使用例

```
> head(d)
  Year mesh      lda        fhc Effort  SD    Route  PG      Area  BC  DA HC x
1 2008    1 4.673596 0.08510851      3   6 4.810494  69 1.0691115  20 107  9 1
2 2008    2 4.117214 0.14009740     48 127 5.086193  43 0.9799341 119  61  9 2
3 2008    3 5.717174 0.06247881      4   3 5.222762 177 1.0937907 170 304 19 3
4 2008    4 4.481805 0.07452234     25  33 5.070376  53 0.8261725  63  88  7 4
5 2008    5 3.653869 0.10567298     18  26 4.984438  25 1.0390412  21  39  4 5
6 2008    6 3.322452 0.09243525     28  15 5.043616  27 1.1372558  29  28  3 1
  y
1 1
2 1
3 1
4 1
5 1
6 2
```

summary() の使用例

```
> summary(d)
      Year           mesh          lda              fhc               Effort
 Min.   :2008   Min.   : 1   Min.   :2.389   Min.   :0.04153   Min.   : 0.0
 1st Qu.:2009   1st Qu.: 7   1st Qu.:3.322   1st Qu.:0.11051   1st Qu.: 5.0
 Median :2010   Median :13   Median :4.108   Median :0.16204   Median :13.0
 Mean   :2010   Mean   :13   Mean   :4.061   Mean   :0.17352   Mean   :18.4
 3rd Qu.:2011   3rd Qu.:19   3rd Qu.:4.606   3rd Qu.:0.22110   3rd Qu.:28.0
 Max.   :2012   Max.   :25   Max.   :5.935   Max.   :0.42741   Max.   :86.0
                                             NA's   :25
       SD             Route              PG              Area
 Min.   :  0.0   Min.   :4.778   Min.   :  4.00   Min.   :0.6389
 1st Qu.:  3.0   1st Qu.:4.924   1st Qu.: 21.00   1st Qu.:0.9310
 Median : 11.0   Median :4.993   Median : 39.00   Median :0.9903
 Mean   : 37.5   Mean   :4.995   Mean   : 58.87   Mean   :0.9967
 3rd Qu.: 29.0   3rd Qu.:5.076   3rd Qu.: 69.00   3rd Qu.:1.0814
 Max.   :571.0   Max.   :5.223   Max.   :302.00   Max.   :1.1896

       BC               DA               HC              x           y
 Min.   :  3.00   Min.   : 11.00   Min.   : 1.00   Min.   :1   Min.   :1
 1st Qu.: 27.00   1st Qu.: 28.00   1st Qu.: 5.00   1st Qu.:2   1st Qu.:2
 Median : 56.00   Median : 61.00   Median : 8.00   Median :3   Median :3
 Mean   : 97.46   Mean   : 86.81   Mean   :15.89   Mean   :3   Mean   :3
 3rd Qu.:108.00   3rd Qu.:100.00   3rd Qu.:19.25   3rd Qu.:4   3rd Qu.:4
 Max.   :720.00   Max.   :378.00   Max.   :82.00   Max.   :5   Max.   :5
                                   NA's   :25
```

2.1.2 **データの構造**

以上の関数を使って表示されるデータは、いわゆる「行列」（縦と横にデータが展開されている）形式になっています。ここで、R でのデータの構造について整理しておきましょう。

まず、1 つ 1 つの値にも形式があります。それは、主に以下のようになっています。

NA はしばしば他の関数を無効化してしまうので注意が必要です（後述）。また、要因は外見は文字列とほぼ同じですが、要因は明確なカテゴリーとして扱われます。外見が数字でも文字列として扱われているケースがあるので注意が必要です。

一方、複数のデータの集まりの形式は、以下のようになっています。

ベクトル 行列などの構造がない、一つながりのデータ。あらゆるデータを含るが、**同一ベクトル**

名称	R での表記	変更方法	例
空値	`NA`		NA
論理値	`logical`	as.logical()	TRUE
実数	`numeric`	as.numeric()	5.3
文字列	`character`	as.character()	`"Tekito"`
要因	`factor`	as.factor()	

名称	R での表記	変更方法
ベクトル	`c`	c()
行列	`matrix`	matrix() または as.matrix()
データフレーム	`data.frame`	data.frame() または as.data.frame()
リスト	`list`	list() または as.list()

内で異なった形式の値を混在させることはできない。無理やり異なった形式の値を混在させると、一番ランクの低い（例えば数字は文字列になれるが、文字列は数字になれない）形式に変更される。例えば、

```
> test <- c(1, 2, "Tekito", TRUE)
> test
[1] "1"      "2"      "Tekito" "TRUE"
> test[1] + test[2] #test の 1 番目と 2 番目の要素を足しなさい
Error in test[1] + test[2] : non-numeric argument to binary operator
> 1 + 2
[1] 3
```

となり、文字列が入っている場合は数字も文字列扱いになってしまう。

行列 ベクトルがいくつも集まり、「行」と「列」という構造を持ったもの。数字以外は含めない。

データフレーム 行列と同じだが、数字以外も使える。R にデータを読ませるときは大概この形式で、実用上最も多用する。

リスト 行数の違うデータだろうがなんだろうが無理やり階層化して含めてしまう（最終奥義）。データの長さや形式がごちゃごちゃなものをとりあえず一つのオブジェクトにまとめたいときに有用。

とりあえずゑくせるなどに打ち込んだデータはデータフレームとして扱うことがほとんどです。今回読み込んだデータもデータフレーム形式です。

2.1.3 行列の一部にアクセスする

行列で表示されているデータは、一部の「行」または「列」のみを取り出すことができます。取り出す方法としては、以下のようなものがあります。

- d[行の番号または名前, 列の番号または名前]
- **d$列名**

```
> d[1, ]
  Year mesh      lda        fhc Effort SD    Route PG     Area BC  DA HC x y
1 2008    1 4.673596 0.08510851      3  6 4.810494 69 1.069111 20 107  9 1 1
> d[, 3]
  [1] 4.673596 4.117214 5.717174 4.481805 3.653869 3.322452 4.206769 3.102632
  [9] 5.360396 4.127018 2.521153 3.451386 3.179972 3.988641 5.291271 4.118098
```

（以下省略）

```
> d$lda #これでも同じ
> d[, "lda"] #これでも同じ
```

#逆に一部のデータを削除する場合は、-をつける

```
> d[, -1]
    mesh      lda        fhc Effort  SD    Route  PG      Area  BC  DA HC x y
1      1 4.673596 0.08510851      3   6 4.810494  69 1.0691115  20 107  9 1 1
2      2 4.117214 0.14009740     48 127 5.086193  43 0.9799341 119  61  9 2 1
```

（以下省略）

これを応用すると、ゑくせるの「オートフィルタ」のように、ある条件に従う部分だけを取り出すということもできます。

#mesh が 100 のところだけ取り出す

```
> d[d$mesh == 100, ]
```

二項演算子

上記の例のように、ある条件に従うかどうかなどを判定させるには、二項演算子を使います。よく使う二項演算子は、以下のようです。

- ==: 等しい
- !=: 等しくない
- &: および
- |: または

2.1.4 基本的な関数

その他、よく使う基本的な関数としては、以下の関数が挙げられます。

`mean()` 列の平均値
`median()` 列の中央値
`sd()` 列の標準偏差
`sum()` 列の値の合計
`min()` 列の最小値
`max()` 列の最大値
`length()` 列のデータ数
`table()` 列の各カテゴリーに含まれるデータ数を示す。2 カテゴリーに拡張した場合は `xtabs()`
`xtabs()` 複数（一つでもいいですが）のカテゴリーごとに、別な変数の合計値を計算する。変数を指定しなければ、カテゴリーの個数となる
`apply()` 全ての列または行に上記の関数を適用させる
`tapply()` ある列に関し、別の列の**カテゴリーごと**に、上記の関数の計算をさせる
`write.csv()` オブジェクトを csv ファイルとして出力する。

いくつかの関数について、使い方を捕捉します。

`table()`

カテゴリーの数を調べるときに使います。

`table(`**カテゴリー変数**`)`

```
> table(d$Year)
2008 2009 2010 2011 2012
  25   25   25   25   25
```

xtabs()

xtabs(**変数 ~ カテゴリー** 1 + **カテゴリー** 2 + ..., **データフレーム**)

#単にカテゴリーの数が知りたい場合は

```
> xtabs(~ Year + mesh, d)
      mesh
Year   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
  2008 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2009 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2010 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2011 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  2012 1 1 1 1 1 1 1 1 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

#ある変数について、カテゴリーごとの合計値が知りたい場合は

```
> xtabs(DA ~ Year + mesh, d)
      mesh
Year     1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  2008 107  61 304  88  39  28  67  22 213  62  12  32  24  54 199  61  61 187
  2009 123  64 348 100  44  33  79  21 218  68  15  28  27  59 220  66  74 187
  2010 130  64 371 115  49  29  79  26 201  63  14  27  25  54 315  68  93 197
  2011 131  63 378 128  47  35  70  28 139  58  12  21  22  50 312  68  84 219
  2012  97  56 369  94  40  36  80  26 111  60  12  16  16  34 309  51  56 192
      mesh
Year    19  20  21  22  23  24  25
  2008 144  56  13  24  35  30  73
  2009 170  65  16  21  36  31  73
  2010 182  57  16  21  40  22  70
  2011 180  70  15  22  41  22  83
  2012 176  62  11  25  35  18  61
```

apply()

データフレームの行、または列ごとに関数を適用させるときに用います。

apply(**データフレーム，1（行の場合）または 2（列の場合），関数**)

```
> apply(d, 2, max)
       Year        mesh         lda         fhc      Effort          SD
2012.000000   25.000000    5.934878          NA   86.000000  571.000000
      Route          PG        Area          BC          DA          HC
   5.222762  302.000000    1.189572  720.000000  378.000000          NA
          x           y
   5.000000    5.000000
#ちょっとトリッキーだが、一気に作図もできる
> par(mfrow=c(3, 5))
> apply(d, 2, hist, main="")
```

tapply()

あるカテゴリーごとに関数を適用させるときに用います。

tapply(**関数を適用させるデータ，カテゴリー，関数**)

```
> tapply(d$DA, d$Year, sum)
2008 2009 2010 2011 2012
1996 2186 2328 2298 2043
```

tapply() は、他にも使いどころがあります。後程紹介します。

2.2 簡単な作図

また、作図もデータの概要をつかむのに有効です。作図は大きく分けると、以下の 2 つに分かれます。

- データを与えれば（決まった形式の）図を作ってくれる「高水準作図」
- 細かい装飾、線や四角などの部品を描画する「低水準作図」

ここでは、高水準作図を中心に紹介します。

2.2.1 散布図

2 つの変数（連続値）の関係を見るときによく使います。

散布図

plot(Y **軸のデータのラベル** ~ x **軸のデータのラベル，オブジェクト名**)

```
> plot(SD ~ Effort, d)
#シンボル（pch）や色（col）を変える場合
#この場合はシンボルを塗りつぶしの丸にし、年毎に色を変える
> plot(SD ~ Effort, pch=16, col=Year, d)
```

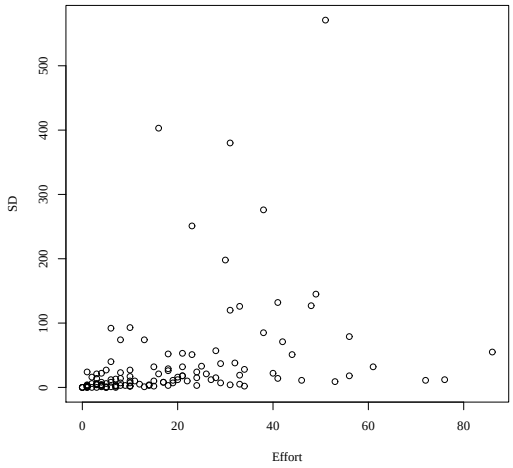

図 1　plot() の例

また、lattice パッケージを使うと、「カテゴリーごとの散布図」を描画することができます。

カテゴリーごとの散布図

xyplot(Y **軸のデータのラベル** ~ x **軸のデータのラベル** | **カテゴリーのラベル，オブジェクト名**)

```
> library(lattice)
> xyplot(SD ~ Effort | mesh, d)
```

2.2.2　シンボルの形と色の指定

これらは本来「低水準作図」と呼ばれるものですが、非常に多用するので、ここで紹介します。

シンボルの色は、col で指定します。指定の仕方は、数字、色の名前、RGB の指定の 3 種類があ

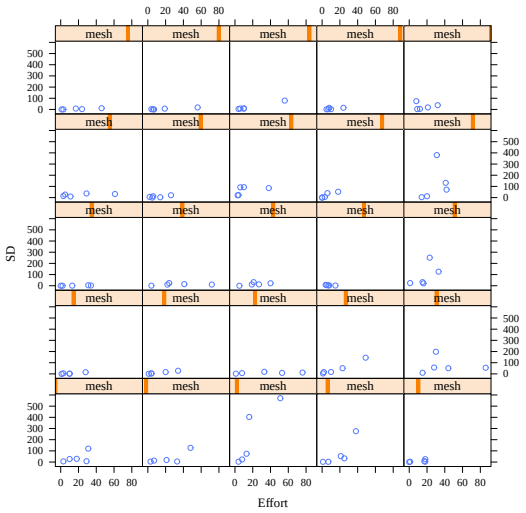

図 2　xyplot() の例

ります。数字ではあまり細かい色まで指定できませんが、数種類を指定するぐらいなら十分です。

　一方、シンボルの形は pch で指定します。指定の仕方は、数字を用います。数字とシンボルの形の関係は、以下のようになっています。

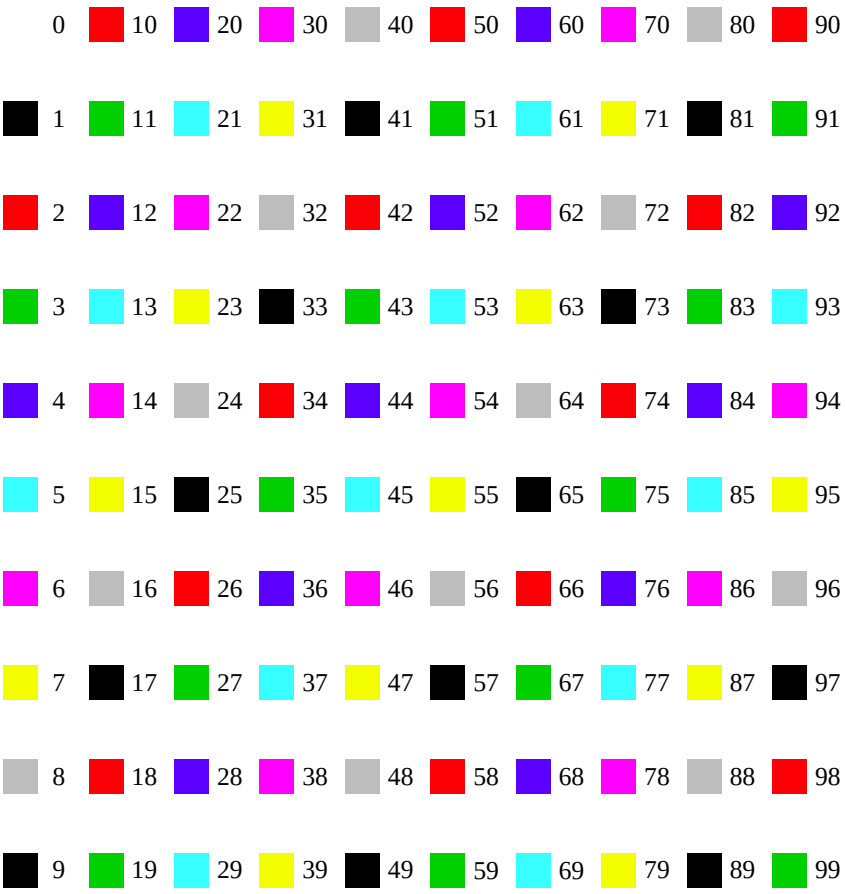

図 3　シンボルの色と指定する数字

□	0	▽	25	2	50	K	75	d	100	}	125		150		175		200		225		250
○	1		26	3	51	L	76	e	101	~	126		151		176		201		226		251
△	2		27	4	52	M	77	f	102		127		152		177		202		227		252
＋	3		28	5	53	N	78	g	103		128		153		178		203		228		253
×	4		29	6	54	O	79	h	104		129		154		179		204		229		
◇	5		30	7	55	P	80	i	105		130		155		180		205		230		
▽	6		31	8	56	Q	81	j	106		131		156		181		206		231		
⊠	7		32	9	57	R	82	k	107		132		157		182		207		232		
✳	8	!	33	:	58	S	83	l	108		133		158		183		208		233		
⊕	9	"	34	;	59	T	84	m	109		134		159		184		209		234		
⊕	10	#	35	<	60	U	85	n	110		135		160		185		210		235		
✡	11	$	36	=	61	V	86	o	111		136		161		186		211		236		
⊞	12	%	37	>	62	W	87	p	112		137		162		187		212		237		
⊠	13	&	38	?	63	X	88	q	113		138		163		188		213		238		
⧈	14	'	39	@	64	Y	89	r	114		139		164		189		214		239		
■	15	(	40	A	65	Z	90	s	115		140		165		190		215		240		
●	16	)	41	B	66	[	91	t	116		141		166		191		216		241		
▲	17	*	42	C	67	\	92	u	117		142		167		192		217		242		
◆	18	+	43	D	68	]	93	v	118		143		168		193		218		243		
●	19	,	44	E	69	^	94	w	119		144		169		194		219		244		
•	20	−	45	F	70	−	95	x	120		145		170		195		220		245		
○	21	·	46	G	71	`	96	y	121		146		171		196		221		246		
□	22	/	47	H	72	a	97	z	122		147		172		197		222		247		
◇	23	0	48	I	73	b	98	{	123		148		173		198		223		248		
△	24	1	49	J	74	c	99	|	124		149		174		199		224		249		

図 4　シンボルの形と指定する数字

2.2.3 箱ひげ図

2 つの変数（片方が連続値でもう片方がカテゴリー）の関係を見るときによく使います。

箱ひげ図

boxplot(Y **軸のデータのラベル** ~ x **軸のデータのラベル，オブジェクト名**)

```
> boxplot(DA ~ Year ,d)
```

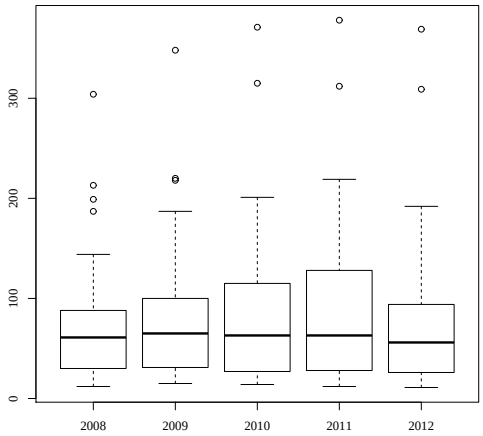

図 5　boxplot() の例

2.2.4 ヒストグラム

1 つの変数の分布をみるときに使います。

ヒストグラム

hist(**ヒストグラムを書きたい変数**)

```
> hist(d$DA)
#区切り幅を変更するときは breaks を指定する
> hist(d$DA, breaks=0:40*10)
```

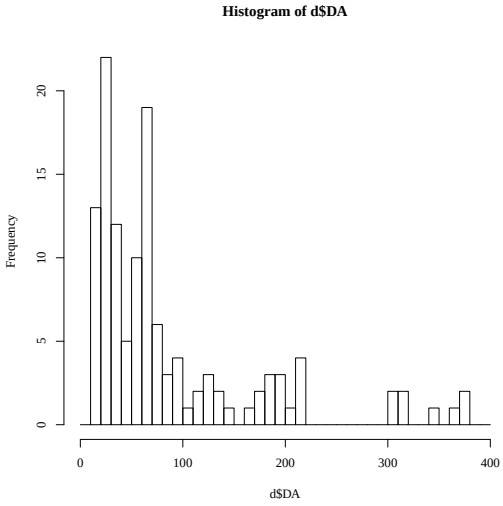

図 6　hist() の例

3 SECOND STAGE

この節では、前節よりも複雑なデータ操作、および作図の技法を扱います。

3.1 データの操作方法

3.1.1 データ同士の結合

データの収集過程で、同じ調査地（場所）について後から別なデータが得られるというのはよくある話です。このような場合、最初にあったデータに後から得られたデータを手で入力して加えていくと、入力ミスなどが起こりえます。このような時は、両方のデータに共通したラベル（例えば調査地名）で合わせて 2 つのデータを結合させることができます。

お渡ししてある other.csv は、各メッシュが属する市町村名、牧草地の有無、最大積雪深が入力されています。これを、元のデータに結合してみましょう。なお、本データは飯島が乱数から生成したもので、実際の市町村の状況とは一切関係ありません^^;)

データ同士の結合 1

merge(**データフレーム** 1, **データフレーム** 2)
データフレームには、共通する「列」が必要
結合させたいデータの列名が違う場合は、merge(..., by.x=, by.y=) **で指定**

```
> e <- read.csv("other.csv")
> f <- merge(d, e)
> head(f,2)
  mesh Year      lda        fhc Effort  SD    Route PG     Area  BC  DA HC x y
1    1 2008 4.673596 0.08510851      3   6 4.810494 69 1.069111  20 107  9 1 1
2    1 2011 4.878003 0.42741105     31 120 5.037620 94 1.144408 365 131 56 1 1
     city G MS
1 Sapporo 1  5
2 Sapporo 1  5
```

また、生データからプロットごとなどに要約値を計算し、それを元データに付与するといったこともよくあります。そのような状況でも merge() は役に立ちます。

データ同士の結合 2

各年で最大のシカ個体数を、各データに付与する。

```
> saidai <- data.frame(tapply(d$DA, d$Year, max))
> saidai$Year <- rownames(saidai)
> colnames(saidai)[1] <- "MaxDA"
> dnew <- merge(d, saidai)
```

3.1.2 データの並べ替え

データの順番や並べ方を変えたいということもよくあります。ここでは、そのような用途で使える関数を紹介します。

- order()
- t()
- reshape()

なお、ここからは表示の都合上、以下のデータフレームを使ってください。

```
> g <- f[, c("Year", "mesh", "DA")]
```

昇順・降順に並べ替え

```
order()
#order() はデータの「順位」を返します

#昇順
> head(g[order(g$DA), ], )
#降順
> head(g[order(g$DA, decreasing=TRUE), ], )
```

さて、上記のコードを見ただけでは、何が行われているのかちょっとわかりにくいです。order()がどのような結果を返すのか見てみましょう。

```
> order(g$DA)
  [1] 104  51  52  55 105  53  54 102  59  64 101 103 120  36  56 106 108  39
 [19]  61 107 117 119  62 109  63 110  37  40  58  65  26  38  60  28 118 116
 [37]  57  27  69  29 111 115  30 113  22  21 112 114  24  25  23  66  78  67
 [55]  68   8  83  96  98  49  70  50   6  76  82 125  48 100  10  46   7   9
 [73]  97  77  31  47  79  80  34  99 123 122 124  84  32  33  35 121  81  16
 [91]  85  17   4  19   1  44  18   5  20   3   2  42  91  92  95  93  94  86
```

```
[109]  88  87  89  71  41  43  45  90  75  11  74  72  73  12  15  14  13
```

この数字は、データの順位を示しています。例えば 1 番目は 104 となっていますが、これは g$DA というデータにおいて、104 番目（行）に入っているデータが最も小さく、次に小さいのは 51 番目に入っているデータということを示しています（decreasing=TRUE を指定した場合は大きい順）。

そして、データフレームは**データフレーム [行, 列]** で指定できることをすでに学びました。よって、上記のコードは、データの順位を調べ、その順番通りに行を表示させる、ということをしています。

行列をひっくり返す

```
t(データフレーム)

> gyaku <- t(g)
```

使いどころとしては、主成分分析など行にこのデータ、列にこのデータにしたいという指定があり、入力していた行列が逆だったときなどです。

最後に、reshape() について説明します。この関数は、データを縦、または横方向に並べ直したいときに使います。ですが、使い方が難しいので、まず実例から入ります。

reshape()（縦方向 → 横方向）

```
> g <- g[order(g$Year:g$mesh), ] #年かつメッシュ順に並べ替え
> wide <- reshape(g, idvar="mesh", timevar="Year",
+                 v.names="DA",direction="wide")
> head(wide)
   mesh DA.2008 DA.2009 DA.2010 DA.2011 DA.2012
1     1     107     123     130     131      97
6     2      61      64      64      63      56
11    3     304     348     371     378     369
16    4      88     100     115     128      94
22    5      39      44      49      47      40
26    6      28      33      29      35      36
```

引数の意味は以下の通りです。

idvar　行になるデータの行名
timevar　列になるデータの列名
v.names　横方向に展開するデータがある列名
direction　展開する方向

reshape()（横方向 → 縦方向）

```
> long <- reshape(wide, varying=list(2:6), times=(2007 + 1:5),
+                 timevar="Year", direction="long")
> head(long)
       mesh Year DA.2008 id
1.2008    1 2008     107  1
2.2008    2 2008      61  2
3.2008    3 2008     304  3
4.2008    4 2008      88  4
5.2008    5 2008      39  5
6.2008    6 2008      28  6
```

引数の意味は以下の通りです。

varying　方向を変更するデータが入っている箇所。なぜか list() で扱うというお作法がある。
times　縦方向に展開したときに、時間を示す列に入る名前。
timevar　時間を示す列の列名。

データを並べる方向ですが、基本的には縦方向です。しかし、時系列のデータなどはしばしば横方向に並べられています。また、時系列のデータを図示するときも、横方向にデータが展開されていた方が描きやすいことがあります（下図参照）。そのような状況で、データの展開方向を変更するときにとても役に立ちます。

時系列データの描画

```
> matplot(t(wide[, -1]), type="l")
```

3.1.3 NA の処理

空欄（欠損値）のことを、R では NA と表記します。こいつが少々曲者です。NA を含むデータフレーム（あるいはベクトル）に上記の関数を適用しても、 **ほとんどは NA が返ってきます**（つまり実行できない！）。しかし、実際のデータには NA が含まれることが多く、NA の処理をきちんとできないと困ることがよくあります。例えば、

```
> d0 <- read.csv("data0.csv")
> head(d0)
  Year mesh      lda         fhc
1 2008    1 4.673596 0.08510851
2 2008    2 4.117214 0.14009740
3 2008    3       NA         NA
```

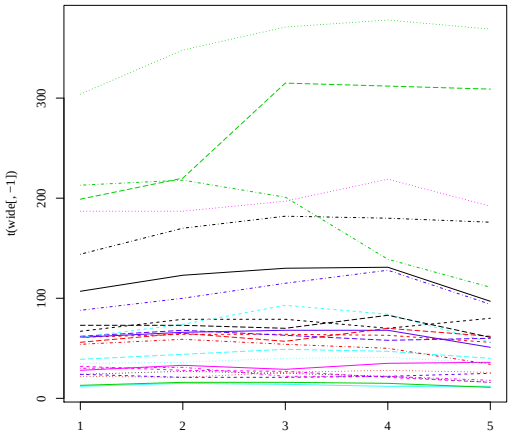

図 7　時系列データの描画例

```
4 2008     4 4.481805 0.07452233
5 2008     5 3.653869 0.10567298
6 2008     6       NA         NA
```

#データに欠損値（NA）が含まれている

```
> mean(d0$lda)
[1] NA
```

#結果も NA になってしまう

そのため、種々の関数を適用する際には、NA を取り除くか、0 などに変換してしまう必要があります。NA に適用できる関数としては、以下のものがあります。

- na.rm=TRUE: 多くの関数の引数として利用可能。NA を含む部分だけ無視して関数を実行する。
- na.omit(): NA を含む行を削除する。
- is.na(): NA であれば TRUE、そうでなければ FALSE を返す。
- ifelse(): NA 用の関数ではないが、NA を処理するのに使える条件分岐関数。

```
> mean(d0$lda, na.rm=TRUE)
[1] 4.0537
```

NA を削除する場合

```
> nrow(d0)
> d02 <- na.omit(d0)
```

```
> nrow(d02)
> summary(d02)
#NA がなくなっている

NA を 0 に変換する場合
> d0$lda <- ifelse(is.na(d0$lda), 0, d0$lda)
#ifelse(条件，TRUE の場合の行動，FALSE の場合の行動)。今回は NA を 0 に置換。
> summary(e) #lda については NA がなくなっている
> nrow(d0) #でもデータ数は減ってない（NA を 0 に変換しただけだから）
```

3.1.4 繰り返し命令

R でデータの操作や作図を行うメリットの一つが、プログラミングによって作業を自動化できる点です。その中でも、繰り返し命令はよく使う物です。繰り返し命令は、

- 同じ作業を違うデータに適用する
- 同じ形式の図を何枚も描く

といった状況で役に立ちます。

繰り返し命令の仕方

```
> for (i in 1:n) {
+ （繰り返させたい作業内容）
+ }
```

さて、これだけでは何のことかさっぱりわからないと思うので、実例を交えて解説します。

例 1：同じ作業を繰り返す

```
> SPUE <- numeric()
> for (i in 1:125) {
+      SPUE[i] <- f[i, "SD"]/f[i, "Effort"]
+ }
```

この例では、SPUE を計算しました。元データには SD（シカ目撃数）と Effort（出猟人日）が入っています。よって、SPUE は SD/Effort になります。そして、データの各行に SD と Effort があるので、SPUE の計算は 1 行目 →2 行目 ... となります。

この作業を、上の例では自動化しています。ポイントは i という文字です。

- 1:125 は、1 から 125 まで 1 ずつ変化する数字の集まり（ベクタ）を作っています。
- i in 1:125 は、この 1 から 125 までの数字が i という文字に入ることを意味します。
- まず、i に 1 が入ります。そして{}内の作業が実行されます。

- 続いて、i に 2 が入ります。そして{}内の作業が実行されます。同じことをしていますが、i は 2 なので、使われるデータが違っています。
- そして、3、4、...125 の順に実行されます。

では、次の例に進みます。今度は、mesh ごとに SD と Effort に相関があるかの検定を、繰り返し命令で実行します。

例 2：同じ作業を繰り返す

```
> f2 <- split(f, f$mesh)
> Nmesh <- nlevels(as.factor(f$mesh))
> pvalue <- numeric()
> r <- numeric()
> for (i in 1:Nmesh) {
+       res <- cor.test(f2[[i]][, "SD"], f2[[i]][, "Effort"])
+       r[i] <- res$estimate
+       pvalue[i] <- res$p.value
+ }
> res <- data.frame(mesh = 1:Nmesh, r, pvalue)
```

今回は、いくつか新しい関数が出てきているので、まずそれを紹介します。

split() データフレームをある列の「カテゴリー」ごとに分割する
as.factor() データを「要因」に変換する
nlevels() カテゴリー変数について、カテゴリー数を計算する
cor.test() 相関の検定を行う関数
numeric() 整数のベクトルを作る

特に、split() は便利な関数です。カテゴリーごとに分割したデータは、[[**数字**]] で取り出すことができます。

上記の例では、データは mesh ごとに分割されています。mesh は 1 から 25 まであります。分割によって、mesh ごとにデータを取り出すことが可能になります。例えば、

```
> f2[[1]]
  mesh Year      lda        fhc Effort  SD    Route PG      Area  BC  DA HC x y
1    1 2008 4.673596 0.08510851      3   6 4.810494 69 1.0691115  20 107  9 1 1
2    1 2011 4.878003 0.42741105     31 120 5.037620 94 1.1444081 365 131 56 1 1
3    1 2010 4.866689 0.18311711     18  29 4.864947 93 0.8543736 107 130 24 1 1
4    1 2012 4.573215         NA     10  27 5.051328 75 0.9347891 118  97 NA 1 1
5    1 2009 4.812471 0.21078215     29   7 5.095017 90 1.0221671 262 123 26 1 1
     city G MS
```

```
1 Sapporo 1  5
2 Sapporo 1  5
3 Sapporo 1  5
4 Sapporo 1  5
5 Sapporo 1  5
> f2[[25]]
    mesh Year      lda        fhc Effort SD    Route PG      Area  BC DA HC x y
121   25 2011 4.423137 0.28788560      8 74 4.986181 40 0.9369917 263 83 24 5 5
122   25 2008 4.291350 0.11415575      9  3 5.076331 40 0.9446751  75 73  8 5 5
123   25 2010 4.247324 0.04453183     32 38 5.095637 47 0.9736861  84 70  3 5 5
124   25 2009 4.292513 0.19178149     21 18 4.887176 35 1.1189972  64 73 14 5 5
125   25 2012 4.110402         NA     12  5 4.898237 42 0.9745762 108 61 NA 5 5
      city G MS
121 Ebetsu 0 72
122 Ebetsu 0 72
123 Ebetsu 0 72
124 Ebetsu 0 72
125 Ebetsu 0 72
```

では次に、繰り返し命令を使って作図をしてみましょう。

例 3：同じ図を何枚でも

```
> Nmesh <- max(f$mesh)
> f2 <- split(f, f$mesh)
> par(mfrow=c(5,5), mar=c(1,1,1,1), oma=c(4,4,0,0), ps=15)
> for (i in 1:Nmesh) {
+      plot(SD ~ Effort, f2[[i]], xlab="", ylab="")
+ }
> mtext(1, line=2, outer=TRUE, text="Year")
> mtext(2, line=2, outer=TRUE, text="Deer abundance")
```

ちなみに、この図を for() を使わないで描画しようとすると、以下のようなコマンドになります（紙の無駄）。

```
> Nmesh <- max(f$mesh)
> f2 <- split(f, f$mesh)
> par(mfrow=c(5,5), mar=c(1,1,1,1), oma=c(4,4,0,0), ps=15)
> plot(SD ~ Effort, f2[[1]], xlab="", ylab="")
```

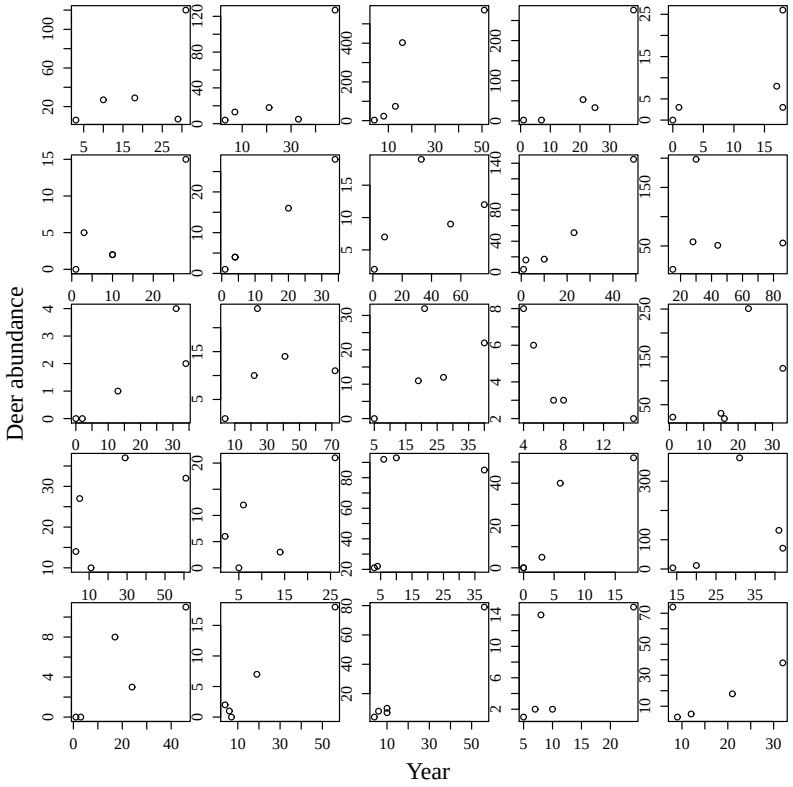

図 8　繰り返し命令による作図

```
> plot(SD ~ Effort, f2[[2]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[3]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[4]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[5]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[6]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[7]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[8]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[9]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[10]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[11]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[12]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[13]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[14]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[15]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[16]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[17]], xlab="", ylab="")
```

```
> plot(SD ~ Effort, f2[[18]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[19]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[20]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[21]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[22]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[23]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[24]], xlab="", ylab="")
> plot(SD ~ Effort, f2[[25]], xlab="", ylab="")
> mtext(1, line=2, outer=TRUE, text="Effort")
> mtext(2, line=2, outer=TRUE, text="Seen deer")
```

コピペで繰り返せば..... と思った方。そこで指導教員（あるいは上司）がこんな悪魔の言葉をささやいたら、どうしますか？

「これじゃ格好悪いんで、図のシンボル変えようか。パソコン使えばすぐでしょ？」

上記の例では作図結果を一つの図にしましたが、場合によっては個別に図を作成したいときもあるでしょう。しかし通常、ファイル名は文字列としてしていますが、文字列を指定するための記号""で囲まれた部分は入力した通りになってしまう（変数として扱えない）ため、そのままでは同じ名前の図がただ上書きされてしまいます。連続で作図し、それぞれに規則性を持った異なるファイル名をつけるためには以下のようにします。

繰り返した結果をそれぞれ別に保存する

（図のウィンドウが立ち上がっている場合は、一度閉じてから実行してください）

```
> file.name <- sprintf("%s.pdf", 1:25)
> for (i in 1:25) {
+      plot(SD ~ Effort, f2[[i]])
+      dev.copy2pdf(file=file.name[i])
+ }
```

3.2 作図

3.2.1 作図の手順

前節では高水準作図を使って、とりあえず図を作りました。しかし、空白や軸など、色々と調整したい点があると思います。まず、思い通りの図を描くために踏むべき、一般的な手順について示します。

- 1 つのウィンドウに図をどう配置するか考える（1 つの図を描くのか、複数の図を描くのか）
- 作図コードを書き込むファイルを用意する（.txt ファイルなど何でもよい）
- （以下の作図に適した形にデータの形を直す）
- 図の形が既存の関数で用意されている場合は、その関数を使って作図する（高水準作図）
- 既存の関数にない形の図を作る、あるいは高水準作図でできた図の装飾をする（低水準作図）
- （ホントはここじゃないんですけど）できあがった図を見て、フォントサイズや色などを調整する

3.2.2 図の配置

R では、1 つのウィンドウが立ち上がってその中に図をどのように配置するか考える必要があります。1 つのウィンドウに 1 つの図しか表示させないなら気にする必要はありませんが、複数の図を配置させる場合は、事前に「何個の図をどのように配置するか」を指定しておく必要があります。指定方法は 2 つ有ります。

- par(mfrow=c(,))
- layout()

mfrow を用いる場合

```
par(mfrow=c(縦に並べる図の数，横に並べる図の数))

> par(mfrow=c(2, 2)) #2 行 2 列（計 4 個）設置すると宣言
> plot(1)
> plot(1)
> plot(1)
> plot(1)
```

普段は 1 枚の図しか描けませんが、これは> par(mfrow=c(1,1)) と指定されているのと同じです。

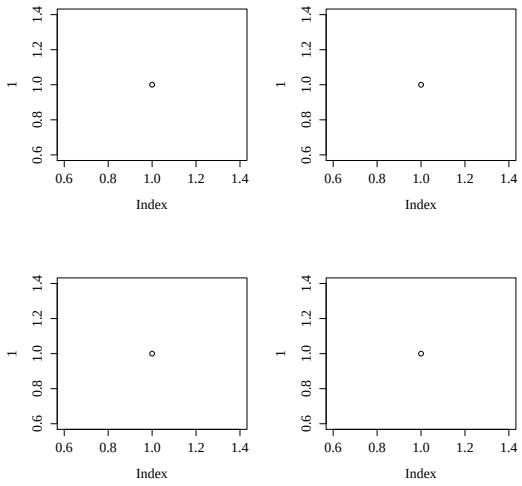

図 9　mfrow() の使用例

layout() を用いる場合

layout(matrix(**図を配置する順番、位置、大きさを示す行列**))

```
> layout(matrix(c(1, 2, 3, 4, 4, 4), ncol=3, byrow=TRUE))
> plot(1)
> plot(1)
> plot(1)
> plot(1)
```

ご覧いただいたらわかるように、4 つめの図が他の 3 つの図を合わせた大きさになっていますね。これと layout() の指定はどのように関係しているのでしょうか？それを見るために、まず中身の部分だけ入力してみましょう。

```
> matrix(c(1,2,3,4,4,4), ncol=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    4    4
```

すると、なんだか図と似たような感じの数字が出てきました。ここでは、行列を作る関数 matrix() を使い、配置する図の順番とその大きさを行列で指定したと言うことになります。

matrix() の使い方ですが、引数の ncol で行数を、byrow=TRUE で「入力したデータを列方向に読んで欲しい」と指定していることになります。

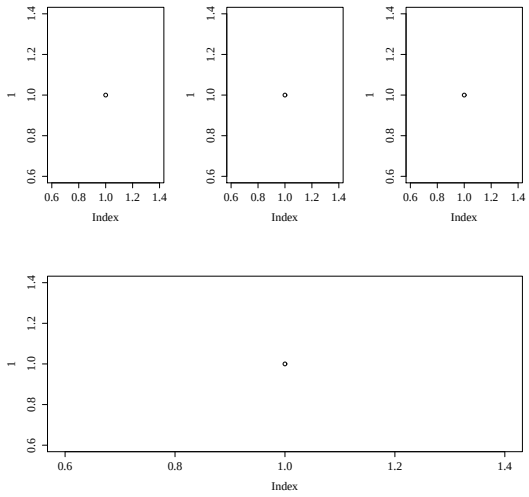

図 10　layout() の使用例

3.2.3　高水準作図関数内での装飾

図の飾り付けは、基本的に低水準作図関数と呼ばれますが、

- タイトルやラベル
- 軸の値や目盛
- 対数軸
- シンボルのサイズ（拡大率）
- 描くシンボルの形や色

などは高水準作図関数内でもある程度調整できます。以下では、散布図中における命令の例を示します。ほとんどは他の高水準作図関数でも応用できます。

タイトル・軸ラベル

```
plot(..., main="タイトルにしたい文字", xlab="x 軸のラベル", ylab="y 軸のラベル", ...)
```

```
> plot(SD ~ Effort, main="SPUE", xlab="Effort (hunter x day)",
+                   ylab="Seen deer)", d)
```

main がタイトル、xlab が x 軸、ylab が y 軸のラベルであり、それぞれ”” の中に文字を入力する。

ラベルで特殊・飾り文字

#字体の変更（例えばイタリック）

```
> expression(italic("ここに斜体にしたい文字"))
```

#数式の記述。

```
> expression(paste(mu,"mol ",m^-2,s^-1))
> expression(paste(CO[2]," concentration (%)"))
```

と書けば、それぞれ μmol $m^{-2}s^{-1}$、CO_2 concentration (%) と表示されます。実際には `xlab=expression(...` といった形で使います。記述部分で" "で囲まれた部分が通常の文字列部分、囲まれていない部分が**数式処理をしている部分**です。paste はこれらを結合する意味を持ちます。

対数軸

```
plot(......, log="x", log="y", log="xy") #それぞれ X 軸、Y 軸、XY 両軸とも対数


> plot(SD ~ Effort, log="x", d)
```

軸の値の範囲・目盛

```
plot(..., xlim=c(x 軸の最小値，x 軸の最大値), ylim=c(y 軸の最小値，y 軸の最大値), ...)


> plot(SD ~ Effort, xlim=c(0, 100), ylim=c(0, 600), d)
```

xlim、ylim でそれぞれ x 軸、y 軸の値の最大値と最小値を調整します。

シンボルの大きさ（拡大率）の指定

```
plot(x,y, cex=*) #cex=*のところの数字を変えると、シンボルの拡大率を指定できる（標準は 1）


実際例は、
> plot(SD ~ Effort, cex=1.5, d)
#ある別のパラメータごとにシンボルのサイズを変える場合は、cex にそのパラメータを入れればよい
> plot(SD ~ Effort, cex=log(DA)-1, d)
```

3.2.4 シンボルの形や色

形については前節を参照して下さい。ここでは、シンボルの色についてもう少し詳しく解説します。

色については、個別の色を指定するほかに、既存の色の組み合わせ（カラーパレット）を使う方法や、同一色で濃淡をつける方法があります。カラーパレットはカテゴリーごとに色を塗り分ける場合、同一色の濃淡は連続的な値と色を対応させるときによく使います。

カラーパレット

rainbow(**作りたい色の数**)　虹のように赤→黄→緑→青→紫と変化する色のセットを作る。
cm.colors(**作りたい色の数**)　水色からピンク色に変化する色のセットを作る

いくつか実例をお見せしたいと思います。

```
> par(mfrow=c(1,2), mar=c(0,0,0,0), oma=c(1,1,1,1))
> pie(rep(1, 12), col=rainbow(12))
> pie(rep(1, 12), col=cm.colors(12))
```

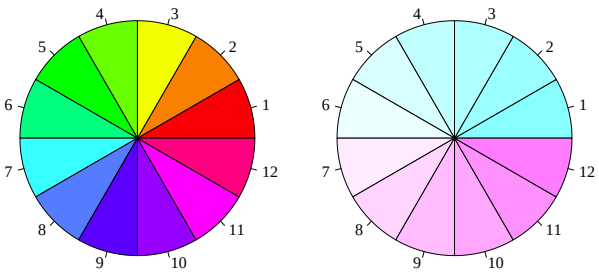

図 11　カラーパレットの例

同一色の濃淡

heat.colors(**作りたい色の数**)　黄色から赤の濃淡。
gray(0〜1 **の値**)　白黒の濃淡。
rgb(,,,0〜1 **の値**)　rgb で指定できるあらゆる色の濃淡

```
> par(mfrow=c(2,2), mar=c(0,0,0,0), oma=c(1,1,1,1))
> pie(rep(1, 12), col=heat.colors(12))
> pie(rep(1, 12), col=gray(1:12/12))
#RGB で明るい緑（0,204,51）を指定
> pie(rep(1, 12), col=rgb(0,204/255,51/255, 1:12/12))
#RGB で濃い青（0,51,102）を指定
> pie(rep(1, 12), col=rgb(0,51/255,102/255, 1:12/12))
```

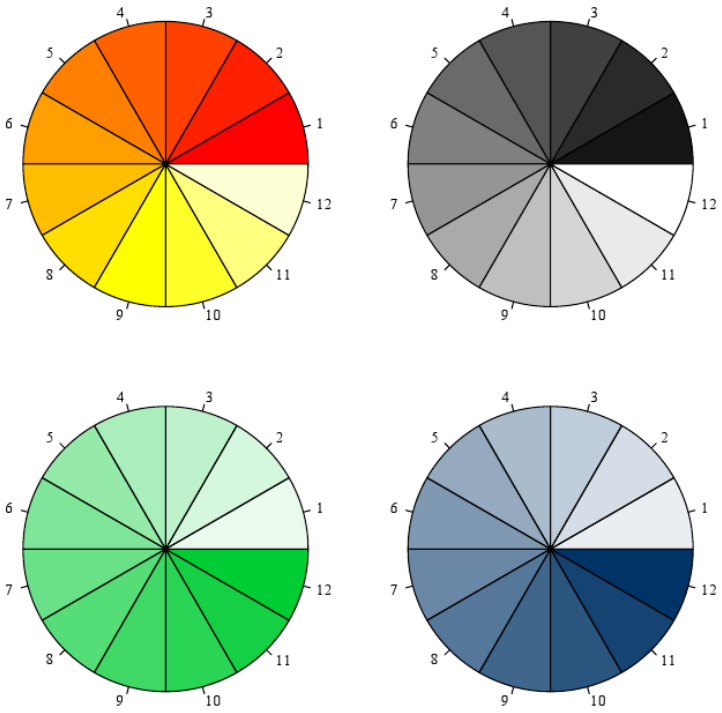

図 12　同一色の濃淡の例

3.2.5 低水準作図

低水準作図は、高水準作図で生成された図の飾り付けが主な役割です。しかし、飾り付けの関数を組み合わせることで、高水準作図では用意されていないような形式の図を作ることもできます。R での作図のキモとなる作図です。

低水準作図として用意されている代表的な機能は、

- 図中に線を引く
- 矢印をつける＋エラーバーをつける
- グリッド線を入れる
- 軸の値の幅や目盛を自分で指定してつける
- 丸（各種シンボル）を描く
- 箱（四角）を描く
- 凡例をつける

などです。

線を引く関数については、

- `lines()`
- `abline()`
- `segments()`
- `curve()`

などが用意されています。それぞれに長所、短所、癖みたいなものがありますので、状況に応じて使い分ければいいと思います。

回帰直線（直線）

`abline(`**線型モデルの結果のオブジェクト**`)`

```
> plot(HC ~ DA, d) #「捕獲個体数-シカ個体数」の関係
> test <- lm(HC ~ DA, d) #回帰直線の切片と傾きを調べる
> abline(test) #回帰分析の結果が入ったオブジェクトを入れればよい
```

`abline()` は別な使い方もできます（後述）。

ただ、`abline()` だと簡単な線しか引けません。線の範囲をデータ範囲に収める、生残率の場合など式の形が線形でない場合などは、`curve()` を使うと便利です。

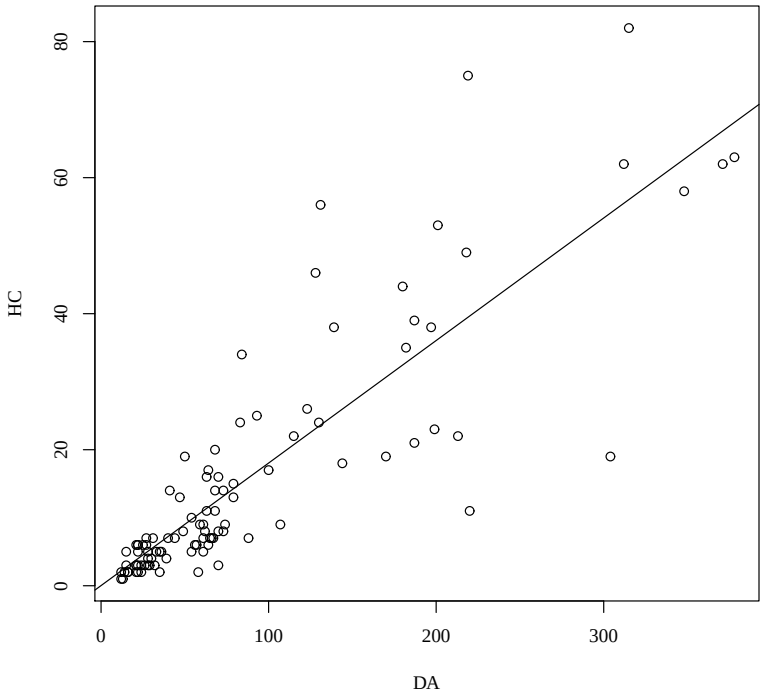

図 13　回帰直線を描画する

回帰直線（複雑な線）

curve(**線を引く式（**x **軸に当たる部分には** x **を入れること）**, from=**, to=**, add=TRUE)
from **と** to **は** x **の最小値と最大値を指定。**

```
> plot(G ~ DA, f[f$Year == 2012, ])
> res <- glm(G ~ DA, family=binomial, f[f$Year == 2012, ])
> co <- res$coefficients #切片と傾きを co に付与
> curve(1/(1 + exp(-(co[1] + co[2] * x))), add=TRUE,
+       from=min(f[f$Year == 2012, "DA"]),
+       to=max(f[f$Year == 2012, "DA"])) #回帰直線を描画
```

co[1] + co[2] * x は予測値そのものですね。add=TRUE というのは、前の図に重ねて線を引くということを宣言するものです。で、x の値の範囲を from と to で決めているわけです。

あと、今回は glm() で誤差構造が binomial、リンク関数が logit を使っていますので、予測値は逆リンク関数である 1/(1+exp(-x)) で変換しています。

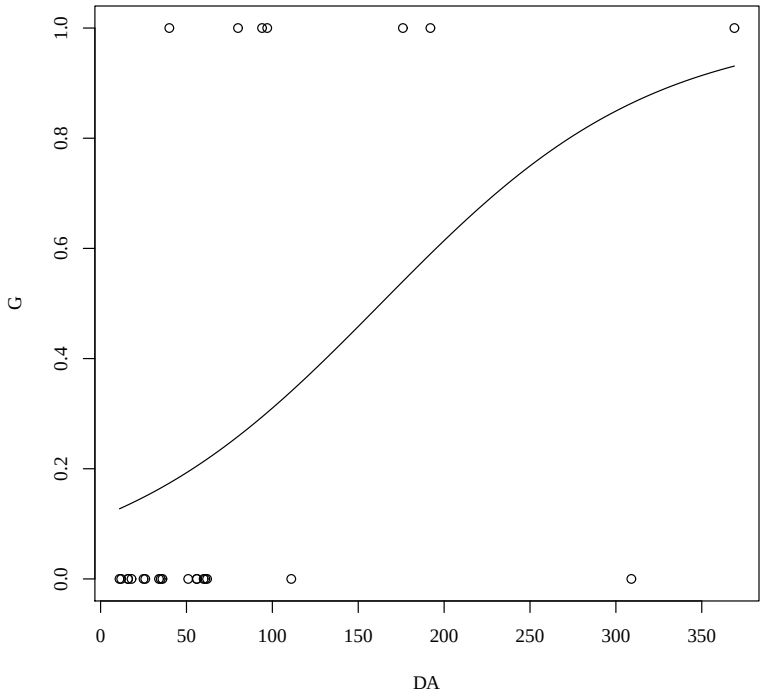

図 14　複雑な線を描画する

線の太さ・種類

線の太さは lwd=、種類は lty=で指定します。線を描く関数の引数として使って下さい。

arrows() を利用したエラーバー

arrows(**始点の** x，**始点の** y，**終点の** x，**終点の** y，length = **矢印の長さ**，
angle = **矢印の角度**，code = **矢印を線のどちら側につけるか**)
code **が** 1 **の場合は始点側に、**2 **の場合は終点側に、**3 **の場合は両方に矢印がつきます**

```
> Ave <- tapply(f$fhc, f$Year, mean)
> SD <- tapply(f$fhc, f$Year, sd)
> barplot(Ave, ylim=c(0, 0.4), xlim=c(0, 4.5))
> arrows(0.7 + 0:3*(1.2), Ave-SD, 0.7 + 0:3*(1.2), Ave+SD, angle=90, code=3)
```

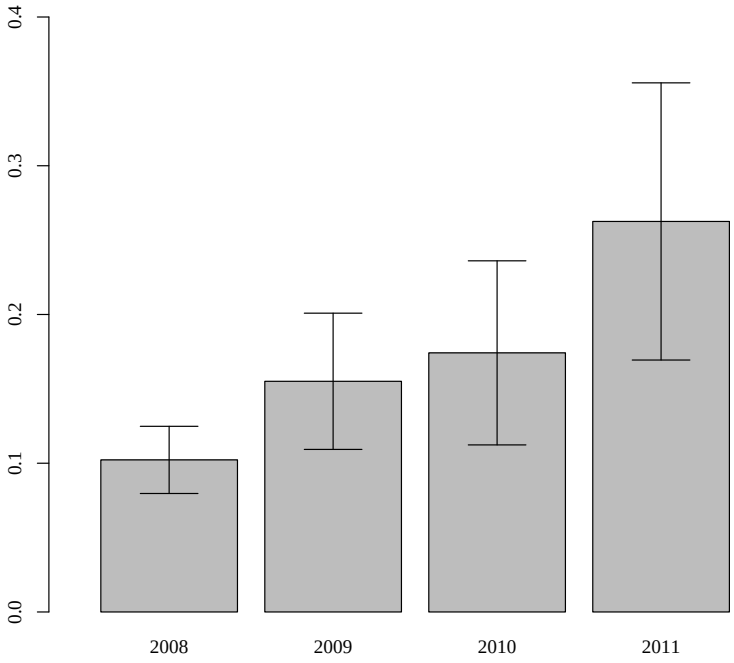

図 15　エラーバーの描画

グリッド線の入れ方

```
> plot(SD ~ Effort, d)
> abline(v = 0:10*10, h=0:6*100, lty=2)
```

v は vetical の意味で、垂直方向に線を引く X 軸上の位置、h は horizontal の意味で、水平方向に線を引く Y 軸上の位置を指定します。引数で lty を指定すれば線の種類も指定できます。

軸の描き方（自分で目盛の幅などを決めたい場合）

```
plot(..., xaxt="n", yaxt="n", ...)
#xaxt、yaxt="n"で x 軸、y 軸の目盛を描かない
axis(軸を描く位置, at=c(自分で打ちたい目盛の値), labels="メモリの値")
#軸を描く位置は数字で指定し、1 は下（X 軸）、2 は左（Y 軸）、3 は上、4 は右

> par(mfrow=c(1,2))
> plot(SD ~ Effort, d)
> plot(SD ~ Effort, xlim=c(0, 100), ylim=c(0, 600), xaxt="n", yaxt="n", d)
> axis(1, at=0:2*50) #X 軸。labels は at と同一なら必要ない
> axis(2, at=0:3*200) #Y 軸。
```

シンボルの描き方

```
points(x 座標, y 座標, pch=シンボルの番号, col="色", type="*")

> plot(SD ~ Effort, f[f$Year == 2008, ], pch=16)
> with(f[f$Year == 2009, ], points(Effort, SD, pch=16, col="red"))
```

引数である type に入れる文字を変えると色々な形式で表示させられます。特に指定しなければシンボルが転々と描かれますが、l とすると線のみが、b とするとシンボルと線が、o とすると b と似ていますがシンボルを貫通した線が描かれます。

箱・グリッドの描き方

```
rect(左の x, 下の y, 右の x, 上の y, col="色", angle=斜線の角度,
     density=斜線の密度 #中を斜線にする場合は
)

> plot(1, type="n", xlim=c(0, 3), ylim=c(0, 3))
> rect(rep(0:2, 3),rep(0:2, each=3),rep(0:2, 3)+1,rep(0:2, each=3)+1, col="red")
```

凡例の書き方

```
legend("場所を指定"または x と y の座標，pch=シンボル番号，col=凡例の色，
       legend="凡例"，（凡例を四角で囲みたくない場合は）bty="n")

実際例は、
> plot(SD ~ Effort, f, pch=16, col=f$Year-2007)
> legend("topleft", pch=16, col=1:5, legend=2008:2012)
#場所の指定方法: "top", "topleft", "bottomright"など
#x と y 座標で指定するなら
> legend(0, 550, pch=16, col=1:5, legend=2008:2012)
```

3.2.6 図の保存の仕方

- Windows: できた図は、図の上で右クリックし、**ビットマップにコピー**を選び、コピーします。そして、ペイントなどに貼り付けて、ファイルとして保存します。
- Windows: 同じ要領で、**メタファイルにコピー**というものがあります。これを実行し、Word などを立ち上げると、図を貼り付けることができます。メタファイルはあまり一般的な形式ではありませんが、拡大縮小に伴う崩れがないのが特徴です（図として保存してもかまいません）。
- PDF や EPS 形式で保存する場合（どの OS でも可能）:

  ```
  > （作図する）
  > dev.copy2pdf(file="ファイルネーム.pdf", family="Times")
  または
  > dev.copy2eps(file="ファイルネーム.eps", family="Times")
  ```

  とします。family はフォントの種類を指定できます。

4 THIRD STAGE

この節では、もう少し複雑なデータの処理、加工方法を扱います。

4.1 配列の扱い

これまでは人間が視覚的に理解できる、2 次元（行列）のデータのみを扱ってきました。しかし、データが大規模になってくると、それ以上の次元のデータを扱う必要があります。このような、n 次元のデータセットは「配列（array)」と呼ばれます。ここでは、配列の扱い方について説明します。

今回はすでに配列を用意してありますので、array.RData をダブルクリックして読み込んでください。arr というオブジェクトが保存されており、これが今回取り扱う配列になります。

4.1.1 配列へのアクセス

配列の場合、データを見たり取り出したりする方法一つ取っても、データフレームとは異なります。

配列の概要をつかむためには、names() や str() を使います。

配列の様子を知る関数

```
> names(arr)
[1] "Doou" "Doto"
> str(arr)
（長いので省略）
```

names() は、配列の最も上位の階層の名前を取り出すことができます。str() は全データの構造を見せてくれるので、どんなデータが入っているかをより細かく見るのに適しています。

今回の配列は、ここまで扱ってきたシカに関するデータが、2 地域分（Doou と Doto）入っているという設定です。

- 第一階層：地域（Doou と Doto）
- 第二階層：シカデータのデータフレーム

では、配列のもっと下の階層まで見てみましょう。配列のある第一階層のみにアクセスする方法としては、以下のような方法があります。

配列の個別要素にアクセス

```
> arr[1]
> arr[[1]]
> arr$Doou
```

最初の 2 つはどう違うのか、ちょっとわかりにくいです。[] はあくまでリストの構成要素としてアクセスするのに対し、[[]] はリストから取り出したデータフレームとして扱っていることを意味します。これまで学んできたような、データフレームを操作する方法は、後者しか受け付けません。

```
> arr[1][, "Year"]
Error in arr[1][, "Year"] : incorrect number of dimensions
> arr[[1]][, "Year"]
  [1] 2008 2011 2010 2012 2009 2008 2010 2012 2009 2011 2008 2009 2011 2010
 [15] 2012 2008 2012 2010 2009 2011 2012 2008 2010 2009 2011 2008 2009 2010
 [29] 2011 2012 2008 2010 2009 2011 2012 2009 2010 2011 2008 2012 2010 2011
 [43] 2008 2012 2009 2010 2009 2008 2011 2012 2011 2008 2010 2009 2012 2011
 [57] 2008 2010 2012 2009 2011 2008 2010 2012 2009 2011 2008 2010 2012 2009
 [71] 2008 2011 2010 2012 2009 2008 2009 2012 2011 2010 2011 2008 2012 2009
 [85] 2010 2008 2012 2009 2010 2011 2008 2009 2011 2010 2012 2008 2009 2010
 [99] 2011 2012 2009 2011 2010 2012 2008 2009 2011 2010 2008 2012 2008 2010
[113] 2009 2011 2012 2009 2011 2008 2010 2012 2011 2008 2010 2009 2012
```

また、配列の第一階層ごとに様子を見たいときには、以下のようにします。

配列の第一階層ごとに表示

```
lapply(配列, head)
lapply(配列, "[", 行, 列) #これでも同じ。"["は行列指定の [] を示す

> lapply(arr, "[", 1:2, )
$Doou
  mesh Year lda   fhc Effort  SD Route PG Area  BC  DA HC x y    city G MS
1    1 2008 4.7 0.085      3   6   4.8 69  1.1  20 107  9 1 1 Sapporo 1  5
2    1 2011 4.9 0.427     31 120   5.0 94  1.1 365 131 56 1 1 Sapporo 1  5

$Doto
  mesh Year lda    fhc Effort SD Route  PG Area  BC  DA HC x y    city G  MS
1    1 2008 5.1 0.0089     17 46   4.9  96  0.9  40 157  1 1 1 Kushiro 1 7.7
2    1 2011 5.3 0.2437      1  5   5.1 193  0.9 116 198 48 1 1 Kushiro 1 7.7
```

lapply() は配列やリストに作用し、第一成階層ごとに指定した関数を適用できる関数です。

4.1.2 配列の加工

アクセスができるようになったところで、今度は配列データを加工してみましょう。

例題 1：第一階層の列ごとの平均値

```
> lapply(arr, mean)
$Doou
   mesh    Year     lda     fhc  Effort      SD   Route      PG    Area
  13.00 2010.00    4.06      NA   18.40   37.50    5.00   58.87    1.00
     BC      DA      HC       x       y    city       G      MS
  97.46   86.81      NA    3.00    3.00      NA    0.28   51.08

$Doto
   mesh    Year     lda     fhc  Effort      SD   Route      PG    Area
  13.00 2010.00    4.28      NA   20.06   43.97    5.00   65.79    1.01
     BC      DA      HC       x       y    city       G      MS
 112.70   96.88      NA    3.00    3.00      NA    0.48    4.10
（以下省略）
```

警告メッセージが出てきますが、これは数字じゃない所に mean() が適用されたことによって出ているので、無視してかまいません。この例でお分かりいただけたように、第一成分ごとに mean() が適用されています。配列に同じ関数をまとめて適用するときに便利です。この作業は、以下のコードと等価です。

```
> for (i in 1:2) {
+      mean(arr[[i]])
+ }
```

例題 2：第一階層の年ごとのシカ個体数の算出

```
> lapply(arr, function(data) {
+ tapply(data$DA, data$Year, sum)
+ }
+ )
$Doou
2008 2009 2010 2011 2012
1996 2186 2328 2298 2043

$Doto
2008 2009 2010 2011 2012
1886 2287 2627 2691 2619
```

tapply() は、カテゴリーごとに関数を適用する関数です。それを、lapply() の中で扱うことで、

第一階層のデータフレームごとに適用することができます。

ここで、似たような別な関数 sapply() を使ってみます。

```
> sapply(arr, function(data) {
+ tapply(data$DA, data$Year, sum)
+ }
+ )
     Doou Doto
2008 1996 1886
2009 2186 2287
2010 2328 2627
2011 2298 2691
2012 2043 2619
```

見た目は lapply() の結果と似ていますが、行列の形になっているように見えます。実は、ここが結果の違いです。lapply() は結果をリストで、sapply() は結果をベクトルまたは行列で返します。一般に、計算結果をさらに利用する場合、sapply() の方が便利なことが多いです。

これまでは、「第一階層」ごとに計算を行ってきました。しかし、場合によってはデータ全体に対して関数を適用したいこともあるでしょう。今回のように、第一階層それぞれはデータフレームという構造を持っている場合、階層ごとに計算を行い、階層ごとの結果を集約するという手順を取ります。

例題 3：データ全体への関数の適用

ここでは、データ全体で「各年の最大のシカ個体数」を求めるということを行います。

```
> MaxDA <- apply(sapply(arr, function(data) {
+                         tapply(data$DA, data$Year, max)
+          } ) , 1, max)
> MaxDA
2008 2009 2010 2011 2012
 304  348  371  378  369
```

関数が階層的になっていますが、順を追ってみていきましょう。まず、sapply() で「第一階層の年ごとのシカの最大個体数」を算出しています。その結果に apply() を 1（すなわち行）方向に適用し、最大値を算出しています。

また、計算した結果を元の配列に加えたいということもあると思います。そのような場合、計算から結果の付与までを lapply() でまとめて実行することができます。

例題 4：例題 2 の結果の付与

ここでは、例題 2 の結果を元のデータに付与してみます。

```
> arr2 <- lapply(arr, function(data) {
+                 temp <- data.frame(tapply(data$DA, data$Year, sum))
+                 temp$Year <- 2008:2012
+                 colnames(temp)[1] <- "MaxDA"
+                 data <- merge(data, temp)
+ }
+ )
> lapply(arr2, "[", 1:2, )
$Doou
  Year mesh      lda        fhc Effort SD    Route PG      Area BC  DA HC x y
1 2008    1 4.673596 0.08510851      3  6 4.810494 69 1.0691115 20 107  9 1 1
2 2008   17 4.108461 0.08992765      5  0 4.777610 39 0.9792562 90  61  5 2 4
      city G MS MaxDA
1  Sapporo 1  5  1996
2 Ishikari 0 58  1996

$Doto
  Year mesh      lda        fhc Effort SD    Route PG      Area BC  DA HC x y
1 2008    1 5.053579 0.00891695     17 46 4.884871 96 0.8957361 40 157  1 1 1
2 2008   17 4.439404 0.06344082     39 39 5.012536 58 1.0602520 42  85  5 2 4
     city G       MS MaxDA
1 Kushiro 1 7.685000  1886
2    Akan 1 4.982628  1886
```

4.1.3 **配列データと作図**

次に、配列データと作図方法について学んでみたいと思います。と言っても、作図の技法そのものは特に違いはありません。作図対象となるデータの取り出し方に、工夫が要ります。

配列データの場合、「第一成分ごと」に同じ図を描きたいことが多いと思います。配列データの第一成分は、名前のほかに数字でも取り出せる（配列のオブジェクト **[[数字]]**）ことを学びました。これを使えば、配列データでも作図ができます。

例題 1：第一階層ごとのシカ個体数と SPUE の関係

```
> par(mfrow=c(1,2))
> for (i in 1:2) {
+      plot(DA ~ I(SD/Effort), arr[[i]], main=names(arr)[i],
+      xlim=c(0, 30), ylim=c(0, 400))
+ }
```

例題 1.5：第一階層ごとのシカ個体数と SPUE の関係

lapply() を使うと、以下のように表現できます。

```
> par(mfrow=c(1,2))
> lapply(arr, function(data) {
+        plot(DA ~ I(SD/Effort), data,
+        xlim=c(0, 30), ylim=c(0, 400)) }
+ )
```

例題 2：SPUE のトレンド

```
> par(mfrow=c(1,2))
> lapply(arr, function(data) {
+             spue <- tapply(data$SD/data$Effort, data$Year,
+                            mean, na.rm=TRUE)
+             plot(2008:2012, spue, type="o", ylim=c(0, 5), cex=2.0, pch=16)
+ }
+ )
```

例題 3：密度指標のトレンド

```
> par(mfrow=c(1,2))
> lapply(arr, function(data) {
+             spue <- tapply(data$SD/data$Effort, data$Year, mean, na.rm=TRUE)
+             spue <- spue/spue[1]
+             pd <- tapply(data$PG/data$Route, data$Year, mean, na.rm=TRUE)
+             pd <- pd/pd[1]
+             bc <- tapply(data$BC/data$Area, data$Year, mean, na.rm=TRUE)
+             bc <- bc/bc[1]
+             plot(2008:2012, spue, type="o", ylim=c(0.5, 2.5), cex=2.0,
+                  pch=16, xlab="Year", ylab="Change of indices from 2008")
+             points(2008:2012, pd, type="o", col="red", pch=16, cex=2.0)
+             points(2008:2012, bc, type="o", col="blue", pch=16, cex=2.0)
+ }
+ )
```

例題 4：データ全体のシカ個体数のトレンド

```
> SumDA <- apply(sapply(arr, function(data) {
+                         tapply(data$DA, data$Year, sum)
+          }), 1, sum)
> plot(2008:2012, SumDA, xlab="Year", ylab="Deer abundance", type="o")
```

4.2 実習 1：複数の図の作成

ここからは実習を通して、より複雑な技術の習得を試みます。まず、シカ密度指標（x 軸）とシカ個体数（y 軸）の関係を作図してみたいと思います。この実習で最終的に出来上がる図は、以下のようになります。

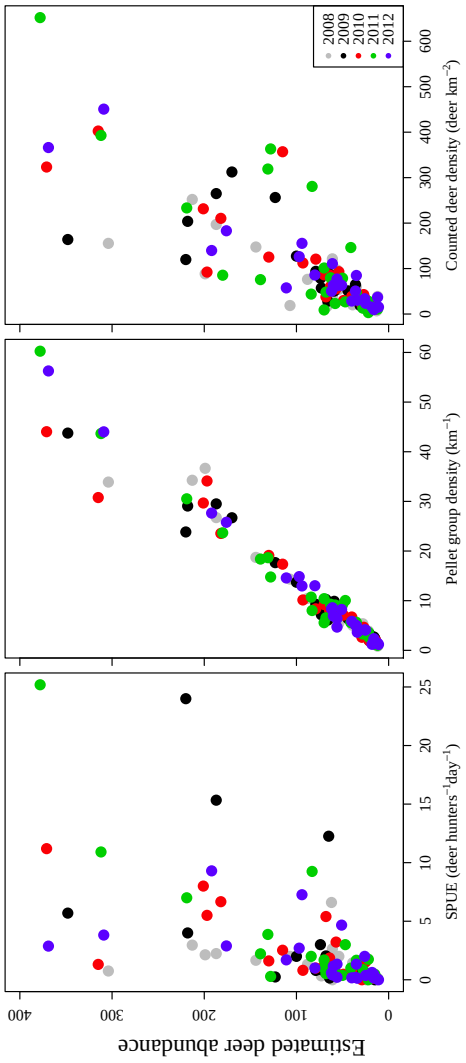
2008
2009
2010
2011
2012
Counted deer density (deer km^{-2})
Pellet group density (km^{-1})
SPUE (deer hunters^{-1}day^{-1})
Estimated deer abundance

図 16　実習 1 の完成図

この図を作るための手順は、大まかには以下のようになります。

- 図の配置を決める
- 余白の調整をする
- 作図を行う。ただし、y 軸の値はすべて同じなので、一番左の図以外は目盛のみとする
- y 軸について、

では、段階を踏んで、作図をしていきます。

図の配置を決める

シカ密度指標は 3 種類（SPUE、糞塊密度、区画法）あります。そのため、今回は横に 3 つの図が並ぶ形にします。

```
> par(mfrow=c(1,3))
```

これで、これから描かれる図は、3 つ横に左から並ぶ形になります。次に、図を描いてみましょう。

図の描画

```
> plot(DA ~ I(SD/Effort), ylim=c(0, 400),
+      pch=16, col=Year,
+      xlab=expression(paste("SPUE (deer ", hunters^{-1}, day^{-1}, ")")),
+      cex=2, d)
> plot(DA ~ I(PG/Route), ylim=c(0, 400),
+      pch=16, col=Year,
+      xlab=expression(paste("Pellet group density (", km^{-1}, ")")),
+      cex=2, d)
> plot(DA ~ I(BC/Area), ylim=c(0, 400),
+      pch=16, col=Year,
+      xlab=expression(paste("Counted deer density (deer ", km^{-2}, ")")),
+      cex=2, d)
> legend("bottomright", pch=16, col=2008:2012,
+        legend=2008:2012)
```

シンボルの色　シンボルの色は、Year 毎に違う色になるようにしています。色は数字でも指定できますので、Year を入れれば自動的に年ごとに違う色になります。

X 軸のラベル　通常の文字と $^{\text{上付文字}}$ を混ぜて使うため、expression(paste()) を使っています。

凡例　引数の col や legend に複数のデータを入れれば、自動的に複数のシンボルが凡例中に描かれます。

さて、出来上がったはいいのですが、完成品と比べると直したいところがあります。

- フォントやシンボルサイズが小さい
- Y 軸は、一番左の図以外は目盛だけにする
- Y 軸のラベルを 1 つだけにする
- 余白を調整する

これらを、調整していきます。

4.2.1 シンボル・フォントサイズの調整

```
> par(ps=15)
> plot(..., cex=2.0)
```

4.2.2 軸の調整

軸を目盛だけにするなどの調整をしたい場合は、自分で軸の指定をする必要があります。高水準作図内では「軸を描かない」という命令を出し、その後低水準作図で軸を描きます。具体的には、以下のようにします。

```
> plot(..., yaxt="n")
> axis(2, at=0:4*100, labels=FALSE)
```

ただし、軸を自分で描画する場合、きちんと値の範囲をそろえておかないと、他の図とずれてしまいます。以下のようにして、値の範囲をそろえておきましょう。

```
> plot(..., ylim=c(0, 400))
```

4.2.3 図の外にラベル

今回の場合は実は必須ではないのですが、複数の図を描くと、その図群全体に対して一つのラベルをつけたい、ということがあります。今回は、Y 軸のラベルがそれに当たります。

このような場合、高水準作図関数でいくら引数を指定しても、高水準作図関数は一つの図を作る関数なので、対応できません。そこで、低水準作図関数で、図群の外側の好きな位置にラベルを入れます。

```
mtext(文字を入れる位置, line=図群からどれだけ離すか, outer=TRUE, text="")
> mtext(2, line=2, outer=TRUE, text="Estimated deer abundance")
```

4.2.4 余白の調整

複数図を作る場合、図の間の間隔を調整する必要があります。調整すべき余白には、実は 2 種類あります。

- 図と図の間の余白
- （先ほどの）図群の外にラベルを描くためのスペース

余白を調整する関数にも、これに対応した 2 種類があります。

`mar()` 個々の図について、図の外の余白を設定する。
`oma()` **複数の図群に関して**、図群の外の余白を設定する。

いずれの関数も、下、左、上、右の順に、取るべき余白の数字を入力します。今回は、以下のようにします。

```
> par(mfrow=c(1,3), mar=c(5,1,0,0), oma=c(0,4,1,1))
```

4.2.5 実習 1 の完成コード

以上の過程をまとめると、以下のようになります。

実習 1 の完成コード

```
> par(mfrow=c(1,3), mar=c(5,1,0,0), oma=c(0,4,1,1), ps=15)
> plot(DA ~ I(SD/Effort),
+      xlab=expression(paste("SPUE (deer ", hunters^{-1}, day^{-1}, ")")),
+      ylim=c(0, 400), ylab="",
+      pch=16, col=Year, cex=2.0, d)
> plot(DA ~ I(PG/Route),
+      xlab=expression(paste("Pellet group density (", km^{-1}, ")")),
+      ylim=c(0, 400), yaxt="n",
+      pch=16, col=Year, cex=2.0, d)
> axis(2, at=0:4*100, labels=FALSE)
> plot(DA ~ I(PG/Route),
+      xlab=expression(paste("Counted deer density (deer ", km^{-2}, ")")),
+      ylim=c(0, 400), yaxt="n",
+      pch=16, col=Year, cex=2.0, d)
> axis(2, at=0:4*100, labels=FALSE)
> legend("bottomright", pch=16, col=2008:2012,
+        legend=2008:2012)
> mtext(2, line=2, outer=TRUE, text="Estimated deer abundance")
```

4.3 実習 2：シカ密度分布図

次は、シカ密度の時系列変化を図示してみたいと思います。この実習で最終的に出来上がる図は、以下のようになります。

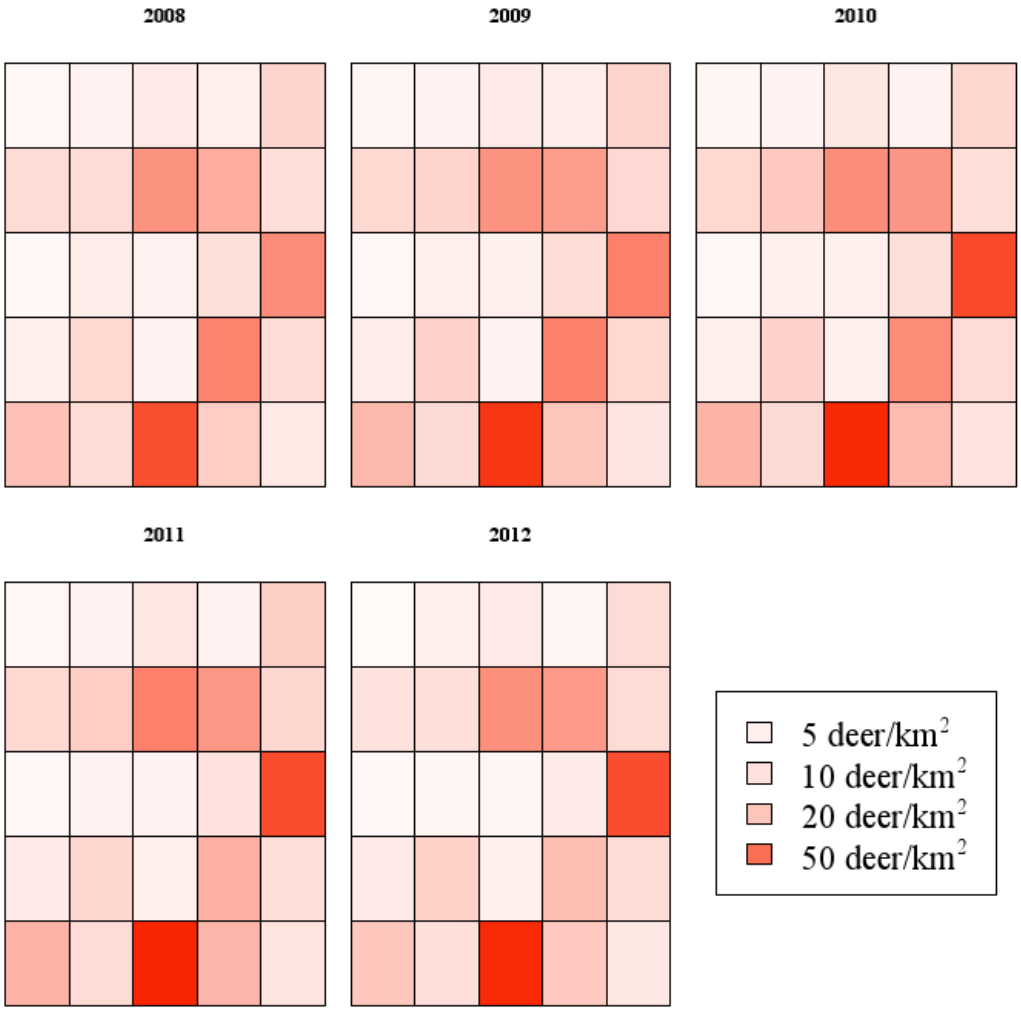
2008
2009
2010
2011
2012
5 deer/km2
10 deer/km2
20 deer/km2
50 deer/km2

図 17　実習 2 の完成図

この図を作るための手順は、大まかには以下のようになります。このような図の高水準作図関数はないので、基本的にすべて低水準作図関数でパーツを作っていくことになります。

- 図の配置を決める
- データを年ごとに分割する
- for() によって、作業の一部を自動化する
- シカ密度が表示されるメッシュを描く
- メッシュごとに、シカ密度に応じて色の濃淡をつける

4.3.1 図の配置

データには 5 年分のデータがあります。そして、判例を 1 つ図とは別に掲載します。そのため、2×3 の配置とします。

```
> par(mfrow=c(2,3))
```

4.3.2 データの分割

年毎に個体数を描くので、データが年ごとになっていないと話になりません。

```
> d2 <- split(d, d$Year)
```

4.3.3 繰り返し命令

```
> Nyear <- nlevels(as.factor(d$Year))
> for (i in 1:Nyear) {
+ （ここに中身を）
+ }
```

4.3.4 メッシュを描画

さて、ここからは for() 内のコマンドを作ることになりますが、for() を使ったプログラムを書くときは、**必ず単純化したプログラムで動作確認をしてから書き始めましょう**。

メッシュを描画する方法は abline() と rect() がありますが、今回はあとでシカ個体数に応じて色分けするので、rect() を使います。

では、まずある 1 年分のデータを描画してみましょう。

```
> plot(1, type="n", axes=F, xlim=c(1,6), ylim=c(1,6),
+      xlab="", ylab="", main=names(d2)[1])
> MaxDA <- max(d$DA)
> with(d2[[1]],
```

```
+ rect(x, y, x+1, y+1, col=rgb(249/255, 37/255, 0, DA/MaxDA)
+ )
+ )
```

最初の plot() では「X 軸と Y 軸の値の範囲を指定し、タイトルだけ出力される空白の図を作図」し、rect() では「シカ密度に応じて色が異なるメッシュを描画」しています。

type=''n'' シンボルなどを描画しない

axes=F 軸を描かない

with() その行だけ、データフレーム名が認識される。そのため、**データフレーム名**$の部分を省略して、列名のみで指定が可能になる。

rgb(**赤, 緑, 青, 透過程度**) RGB で色の指定ができる関数。ただし、通常 RGB は 0〜255 で赤、緑、青の色の指定をするが、この関数内ではすべて 0〜1 の間に値が収まっている必要があるので、255 で割る。透過程度も 0〜1 で指定。今回は、最大のシカ個体数を分母にしている。

4.3.5 凡例

凡例は、通常は「図の内側」に描画する物です。しかし、完成図を見ると、凡例は独立した一つの位置にあります。これは、「空白の図を出力して、その中に凡例だけを描く」ということをしています。その方法は、以下のようになります。

```
> plot(1, type="n", axes=F, ann=F, xlim=c(1, 6), ylim=c(1, 6))
> legend("center", col="black",
+        fil=rgb(249/255, 37/255,0, c(25, 50, 100, 250)/MaxDA),
+        legend=c(expression(paste("5 deer/", km^{2})),
+                 expression(paste("10 deer/", km^{2})),
+                 expression(paste("20 deer/", km^{2})),
+                 expression(paste("50 deer/", km^{2}))),
+        cex=2.0
+ )
```

4.3.6 実習 2 の完成コード

以上の過程をまとめると、以下のようになります。

実習 2 の完成コード

```
> d2 <- split(d, d$Year)
> N.year <- nlevels(as.factor(d$Year))
> MaxDA <- max(d$DA)
> par(mfrow=c(2, 3), mar=c(0,0,3,0), oma=c(1,1,1,1))
> for(i in 1:N.year) {
+     plot(1, type="n", axes=F, xlim=c(1,6), ylim=c(1,6),
+          main=names(d2)[i])
+     with(d2[[i]],
+          rect(x, y, x+1, y+1, col=rgb(249/255, 37/255, 0, DA/MaxDA))
+     )
+ }
> plot(1, type="n", axes=F, ann=F, xlim=c(1, 6), ylim=c(1, 6))
> legend("center", col="black",
+        fil=rgb(249/255, 37/255,0, c(25, 50, 100, 250)/MaxDA),
+        legend=c(expression(paste("5 deer/", km^{2})),
+                 expression(paste("10 deer/", km^{2})),
+                 expression(paste("20 deer/", km^{2})),
+                 expression(paste("50 deer/", km^{2}))),
+        cex=2.0
+ )
```